Пирогов В.Ю.

ASSEMBLER

Учебный курс

Москва
Издатель Молгачева С.В.
2001

УДК 004.438 ASSEMBLER
ББК 32.973.26-018.1
П 33

Пирогов В.Ю.
ASSEMBLER. Учебный курс.
- М.: Издатель Молгачева С.В., Издательство Нолидж, 2001. - 848 **с.,** ил.

Издатель Молгачева С.В.
Лицензия ИД № 03567 от 19 декабря 2000 г.

Издательство Нолидж
Лицензия ЛР № 064480 от 04 марта 1996 г.

ISBN 5-89251-101-4

Содержание

Аннотация.

Глава 1. Трансляция программ с языка ассемблера.

В главе даны начальные сведения о средствах программирования на языке ассемблера. Приводятся простейшие программы.

Глава 2. Посвящена адресному пространству и структуре программ.

Анализируется адресное пространство компьютеров IBM PC, работающих под управлением ОС MS DOS.

Рассматривается структура программ на языке ассемблера.

Глава 3. Первые программы.

Примеры и подробный анализ программ в части ввода-вывода информации, а также понятие стека.

Глава 4. Обзор команд микропроцессоров 8088/8086.

В этой главе приведен список команд микропроцессора 8088/8086, программы и фрагменты программ на ассемблере.

Глава 5. Работа микропроцессора в защищенном режиме.

Приводится описание функционирования процессора в защищенном режиме и основные понятия защищенного режима и алгоритм перехода.

Глава 6. Уровни программирования.

Даны примеры программирования внешних устройств компьютера при помощи функций DOS, функций BIOS и путем прямого обращения к устройству.

Глава 7. Клавиатура, дисплей, принтер.

Приведено более подробное описание программирования трех названых устройств.

Глава 8. Работа с файлами.

Дано подробное описание средств MS DOS для работы с файлами. Рассмотрены обработка различного вида файлов, перенаправление ввода-вывода, одновременное открытие большого числа файлов.

Глава 9. Прерывания.

Рассмотрены следующие вопросы:

Аппаратные и программные прерывания, перехват прерываний, контроллер прерываний и его программирование.

Глава 10. Графический вывод.

Дано введение в графическое программирование VGA на примере одного из графических режимов.

Глава 11. Работа с памятью.

Рассмотрены вопросы: средства MS DOS управления памятью, программный запуск программ, оверлеи.

Глава 12. TSR-программы.

Рассмотрены все аспекты создания резидентных программ: перехват прерываний, неинтерабильность, разрешение конфликтов и др.

Глава 13. Модульное программирование.

Рассмотрены вопросы написания программ, состоящих из нескольких модулей, а также проблема передачи параметров.

Глава 14. Структура информации на диске.

Подробно разбирается структура информации на диске: каталоги, FAT-таблицы, таблицы параметров, структура EXE-файлов для MS DOS и Windows и т.п.

Глава 15. Языки высокого уровня.

Рассмотрены вопросы интерфейса языков высокого уровня (Паскаль, Си, Basic) с ассемблером. Рассмотрены некоторые вопросы программирования на языках высокого уровня в свете эффективности получаемого кода.

Глава 16. Загружаемые драйверы.

Излагается теория написания и структура загружаемых драйверов для MS DOS.

Глава 17. Работа с "мышью" на языке ассемблера.

Подробно описано программное управление манипулятором "мышь" посредством стандартного драйвера.

Глава 18. Элементы теории вирусов.

Рассмотрены проблемы борьбы с компьютерными вирусами.

Глава 19. Проблемы компьютерной безопасности.

Рассмотрены проблемы компьютерной безопасности и, в частности, защиты программного обеспечения от несанкционированного использования.

Глава 20. Микропроцессоры 8088/8086.., 80486...

Дан сравнительный анализ развития семейства микропроцессоров Intel, с точки зрения программиста. Дается также описание и примеры программирования арифметического сопроцессора.

Глава 21. Программирование в локальных сетях.

Описаны средства написания программ, работающих в локальных сетях. Рассматривается локальная сеть под управлением Novel NetWare. Подробно описаны протоколы IPX, SPX.

Глава 22. Здесь рассказывается о том, какая в MS DOS имеется память и как ее использовать.

Описаны способы программного использования различных видов памяти в среде MS DOS (расширенная, дополнительная, верхняя).

Глава 23. Тестирование оборудования.

Приведено несколько примеров тестирования оборудования.

Глава 24. Начала программирования для WINDOWS.

Рассматривается программирование в среде WINDOWS в 16-битном варианте.

Глава 25. 32-битное программирование для Windows.

Рассмотрено программирование для операционных систем Windows 95, 98. Рассматривается консольный режим, использование ресурсов.

Глава 26. Программирование в защищенном режиме.

Рассматривается защищенный режим, приводится пример программирования в защищенном режиме с обработкой исключений и прерываний.

Глава 27. Программирование VGA адаптеров.

Дано описание средств программирования VGA-адаптеров, включая программирование нестандартных режимов. Рассматривается также программирование SVGA-адаптеров, VESA стандарт.

Предисловие.

Во всякой книге предисловие есть первая и вместе с тем последняя вещь: оно или служит объяснением цели сочинения, или оправданием и ответом на критику.

М.Ю.Лермонтов
Герой нашего времени.

Латынь из моды вышла ныне.

А.С.Пушкин
Евгений Онегин.

История создания этой книги состоит из нескольких этапов. Вначале это были отдельные главы по некоторым вопросам языка ассемблера для IBM PC. Я объединил их в одну книгу. Однако мне хотелось отойти от некоторых стереотипов написания подобных книг. Вот эти стереотипы:

1. Использование большого количества фрагментов, но не готовых программ.

2. Упор на начинающих программистов. Разбираются лишь простые вопросы программирования.

3. Ограниченный объем Справочного материала.

Я решил написать книгу, которая бы была полезна как для начинающих, так и для профессиональных программистов. Одновременно книга должна была быть такой, чтобы читатель мог найти в ней всю (или почти всю) необходимую справочную информацию. Именно о такой книге я сам мечтал, когда делал свои первые шаги в программировании. "Но ведь такая книга будет очень толстой", - сказал мне внутренний голос. "Ну и что?" - сказал другой - "Зато в ней будет большое количество материала, множество программ. И в конце концов такой книги еще не было?".

Вот такую книгу я попытался сделать. Насколько она соответствует изложенным выше принципам - Судить Вам. Вы найдете в ней большое количество (около 180) небольших, но рабочих (!) программ. Каждая из этих программ демонстрирует каки-либо особенности. Разобравшись, Вы легко сможете взять на вооружение тот подход, который в ней применялся.

В книге Вы встретите изрядное количество "ссылок вперед" - программисты знают, что это такое. Я думаю, что это не вредно. Программирование - не математика, и применять для его изложения аксиоматический метод не стоит. Здесь важно идти от практики. Программисты знают, что при изучении любого языка программирования желательно узнать Два-три оператора и тут же написать простую программу - после этого все дело сдвинется с мертвой точки.

Зачем нужен язык ассемблера? - спросят меня. Самой простой и убедительный ответ на поставленный вопрос такой: "Затем, что это язык процессора и, следователь-

но, он будет нужен до тех пор, пока будут существовать процессоры". Более пространный ответ на данный вопрос содержал бы в себе рассуждение о том, что ассемблер может понадобиться для оптимизации кода программ, написания драйверов, трансляторов, программирования некоторых внешних устройств и т.д. Для себя я, однако, припас другой ответ: программирование на ассемблере дает ощущение власти над компьютером, а жажда власти — один из сильнейших инстинктов человека. И еще: язык ассемблера - язык универсальный. Действительно, компьютеры на базе микропроцессоров Intel широко распространились по всему свету. И какая бы операционная система ни работала на компьютере - OS/2, Windows, Unix или старенькая MS DOS, язык останется тем же, изменится лишь взаимодействие с операционной системой.

Я не ставил своей целью изложить какие-то строгие правила программирования. Для языка ассемблера это вообще не характерно. Но Вы найдете здесь большое количество программ, которые работают и которые можно дополнять и модифицировать по своему желанию. Все эти программы написаны мной, но все они Ваши. Весь материал, собранный мной из книг, файлов и из собственного опыта также принадлежит Вам.

Вначале книга была посвящена программированию только в среде MS DOS. По сравнению с первым вариантом объем книги вырос почти вдвое. Я добавил главы, посвященные программированию в среде Windows. Кроме того, в других главах сделаны добавления, касающиеся Windows. Подробно рассматривается в книге и защищенный режим. Возможно, в будущем книга охватит и другие операционные системы.

О моей работе знали практически все программисты моего родного города Шадринска. Мне хотелось бы назвать несколько имен. Эти люди поддерживали меня. На их мнение и советы я опирался, а некоторые предоставляли мне свои материалы. Это Кудрявцев А., Шохирев М., Кияшко И., Слинкин Д., Галишев П., Иванов Д., Эминов Р. и др. Особенно я хочу поблагодарить мою жену за то, что поддерживала и верила в меня в течение всей работы над книгой.

Вступление.

Послушайте, ребята, Что вам расскажет дед.

А.К. Толстой.
История государства Российского от Гостомысла до Тимашева.

Для того чтобы приступить к изучению данной книги читатель должен иметь хотя бы минимум предварительных знаний. Что же нужно знать и уметь?

1. Уметь работать с числами в различных системах счисления (двоичной, десятичной, шестнадцатеричной). Впрочем, при программировании на ассемблере такая сноровка быстро приобретается. Уметь работать с числами на битовом уровне - без этого нельзя понять, как работают сдвиговые и логические операции, как процессор работает с отрицательными числами.

2. Совершенно необходимы некоторые навыки программирования, так как я не уделяю много внимания алгоритмической стороне проблемы. Такие навыки быстро приобретаются при программировании на одном из языков высокого уровня. Язык ассемблера слишком детален, чтобы начинать программировать именно с него. По своей идеологии язык ассемблера больше всего напоминает язык BASIC (в старом, DOS'овском его варианте).

3. Иметь навыки работы в среде операционных системы MS DOS и Windows - знание команд, структуры файловой системы и т.д. Иметь хотя бы некоторое представление о работе со строковыми компиляторами и о командной строке.

В конце книги я привожу достаточно полный список компьютерной литературы, к которой Вы можете обратиться, если моя книга не оправдает почему-либо Ваших надежд.

Книга состоит из 27 глав и 9 приложений. В приложения я вынес, во-первых, справочный материал, во-вторых, тот материал, который по каким-либо соображениям не нашел свое место в главах. Возможно, в будущем некоторые из этих приложений также превратятся в главы.

Книга писалась в два этапа, и это наложило на нее определенный отпечаток. Некоторые главы были оставлены мною почти без изменения. Так осталась глава 10, в которой фактически излагается программирование 10h-ого графического режима. Поскольку, однако, все, что изложено в ней, остается справедливым и для VGA, я ничего не стал менять, добавив лишь новую главу, где VGA и SVGA-адаптеры рассматриваются более подробно и дается большой справочный материал по регистрам видеосистемы. То же можно сказать и о главе 5. Она была посвящена микропроцессору 80286. Я внес в нее лишь незначительные изменения, но добавил две главы. Одна из них посвящена программированию в защищенном режиме, а во второй дана сравнительная характеристика микропроцессоров семейства Intel. В этих главах рассказывается, в частности, о новых возможностях микропроцессоров 80386, 80486, Pentium.

Вместе с тем в книгу не вошел материал, устаревший явно. К такому материалу я отношу в первую очередь программирование давно устаревших видеосистем. Конеч-

но, мне могут возразить, что видеорежимы остались. Однако ими уже никто не пользуется, так что не стоит о них и говорить.

Работая с моей книгой, Вы обратите внимание, что часть справочного материала вынесена в приложения, а часть содержится непосредственно в главах. Принцип здесь такой: в приложения я старался выносить ту информацию, которая может понадобиться Вам при работе с разными главами. Например, в главе, посвященной программированию мыши, содержится вся справочная информация по функциям драйвера мыши, поскольку для понимания других глав эта информация не нужна. Здесь опять напрашивается аналогия из программирования - можно пользоваться процедурами, а можно и макроопределениями. Некоторый материал вынесен мною в приложения, по причине его небольшого объема и отсутствия связи с материалом в других главах и приложениях.

Те, кто уже знаком с языком ассемблера, может читать книгу в любом порядке. Начинающим же программистам я бы посоветовал первые восемь глав изучать по порядку, а далее действовать по своему усмотрению.

Как я уже упоминал, все приведенные в книге программы написаны мной. Лишь в некоторых случаях я писал программы, скажем так, по мотивам программ других авторов, используя некоторые их идеи. Но в этом случае я делаю специальные оговорки, ссылаясь на другие книги и авторов.

О терминологии.

Термины "транслятор", "транслирование" в данной книге являются синонимами терминов "ассемблер", "ассемблирование". Кроме того, термин "ассемблер" используется нами как синоним термина "язык ассемблера".

Средства трансляции.

Большая часть программ в данной книге рассчитана на работу с транслятором ассемблера фирмы Микрософт MASM.EXE версии 5.0 (или даже ниже) и редактора связей той же фирмы LINK.EXE (версии 4.0). Программа EXE2BIN.EXE[1], используемая для старых версий LINK, чтобы преобразовывать программы формата EXE в программы формата COM, теперь не нужна, т.к. программа LINK.EXE теперь сама делает всю работу. Те случаи, когда нам понадобятся программы трансляции более высоких версий (MASM версии 6.0, LINK версии 5.0), мы оговорим особо. В отдельных случаях нам понадобятся трансляторы TASM.EXE и TLINK.EXE (TASM32.EXE, TLINK32.EXE), но и об этом читатель будет своевременно информирован.

---

[1] Существовало довольно много заменителей программы EXE2BIN.EXE, но это уже история.

Книгу посвящаю моей жене Люде.

Глава 1. Трансляция программ с языка ассемблера.

Исполнение предприятия приятно щекочет самолюбие.

Козьма Прутков.

Язык ассемблера фактически представляет собой машинный язык (язык процессора), где коды команд заменены именами. Человек лучше ориентируется в именах, чем в числах, поэтому язык ассемблера проще для понимания, чем машинный язык. Кроме того, сами имена могут быть говорящими, например, MOV (от MOVE - перемещать), ADD (прибавлять) и т.п., что дает дополнительные удобства. Другим упрощением языка ассемблера по отношению к машинному является использование меток вместо конкретных адресов. Это значительно упрощает работу, т.к. не нужно думать, по какому адресу расположена та или иная команда или данные. Вот два основных момента, которые определяют язык ассемблера и отличают его от машинного языка. Дальнейшее развитие ассемблера шло по пути совершенствования макросредств. Ассемблер, имеющий в своем распоряжении макросредства, называют макроассемблером. Я в своей книге почти не буду касаться макросредств, т.к. использование их, на мой взгляд, не слишком облегчает программирование на ассемблере, а, скорее, скрывает некоторые важные моменты. Во всяком случае, я не советовал бы использовать их начинающим: детальное проникновение во все тонкости работы программы поможет Вам в будущем стать профессиональным программистом.

Для того чтобы начать работать на языке ассемблера, создадим в одном из разделов жесткого диска подкаталог ASM. Скопируем в него следующие файлы: MASM.EXE - транслятор с языка ассемблера, который преобразует исходный текст программы в объектный файл (PRIMER.ASM --> PRIMER.OBJ), LINK.EXE - компоновщик (редактор связей)[2], преобразующий объектный файл в исполняемую программу, которая может быть запущена из операционной системы, редактор текста, с помощью которого мы будем подготавливать тексты программ, что-нибудь попроще, EXE2BIN.EXE - данная программа позволяет преобразовывать некоторые EXE-программы в COM-программы[3] и может пригодиться, если у Вас старая версия программы LINK.EXE, DEBUG.EXE - отладчик для просмотра и корректировки EXE и COM - программ. При работе с

---

[2] LINK в переводе с английского означает "связывать". Поэтому название "редактор связей" для программы LINK.EXE будет более правильным и более понятным. Слова "компоновщик" и "компоновка" пришли из жаргона программистов.

[3] В версиях DOS 6.00 и выше утилита EXE2BIN.EXE исчезла. Зато у программы LINK.EXE появился ключ /TINY (можно /T), позволяющий делать COM-программы сразу из объектного модуля. Следует также заметить, что программа TLINK.EXE изначально имела такую возможность - ключ /T. Т.о., чтобы получить COM-программу из объектного модуля P.OBJ, достаточно набрать строку: LINK P /TINY; или в сокращенном варианте LINK P /T;. Существуют, однако, программы заменители EXE2BIN, например, программа EXE2COM.

информацией на диске нам также может оказаться полезным какой-нибудь дисковый редактор, например, программа DISKEDIT.EXE из пакета Norton Utilites.

Вместо программ MASM.EXE и LINK.EXE производства фирмы Microsoft можно с тем же успехом использовать турбо-ассемблер TASM.EXE и турбо-компоновщик TLINK.EXE производства фирмы Borland. Некоторые особенности этих утилит и их отличие от аналогичных программ фирмы MicroSoft пока для нас несущественны, однако, работа с командной строкой у Турбо-Ассемблера и Турбо-Компоновщика иная[4], чем у Микрософтовских программ.

Итак, все готово. Не беда, что Вы еще не знаете языка. Первые шаги делайте вместе с нами. Наберите в редакторе следующий текст, который являет собой нашу первую программу.

```
;-----------------------------------------------------------
TITLE  PRIMER11
;--вывод строки символов в текущую позицию курсора—
CODSEG  SEGMENT
ASSUME CS:CODSEG, DS:CODSEG, SS:CODSEG, ES:CODSEG
ORG 100H
BEGIN:                        ;метка
     JMP BEG_CODE             ;безусловный переход
;текст для вывода на экран
TEXT     DB 'Это моя первая программа на языке ассемблера. $'
;$ - признак конца строки
BEG_CODE:                     ;метка начала основного входа
;две следующие строки необходимы для правильной работы
;EXE-программы, для COM-программы их можно опустить
     MOV AX,CS                ;содержимое регистра CS
     MOV DS,AX                ;пересылаем в регистр DS
;--вывод строки--
     LEA DX,TEXT              ;где находится строка
     MOV AH,9                 ;номер функции DOS
     INT 21H                  ;вызов функции
;----теперь выходим в операционную систему———
     MOV AH,4CH               ;номер функции DOS
     INT 21H                  ;вызов функции
CODSEG   ENDS                 ;конец сегмента
     END BEGIN                ;конец программы и точка входа
```

Рис. 1.1. Вывод текста в текущее положение курсора. Программа будет работать, как в EXE-, так и COM-варианте

---

[4] Поддержка синтаксиса MASM в турбо-ассемблере осуществляется в так называемом режиме Ideal, в котором синтаксис MASM поддерживается в более простом варианте. По умолчанию же турбо-ассемблер поддерживает полный стандартный синтаксис MASM.

Программа мной довольно полно прокомментирована. Вы, наверное, уже догадались, что комментарии в программах на языке ассемблера пишутся после символа ';' - точка с запятой. Еще раз призываю Вас не пугаться, а принять программу как данное. Запишем ее на диск под именем PRIMER11.ASM.

Теперь запустим программу MASM.EXE. На экране появится строка Source filename [.ASM]:, набираем в этой строке имя нашей программы, т.е. PRIMER11 (расширение можно опустить) и нажмем клавишу ENTER. Далее все до предела просто - отвечаем нажатием на клавишу ENTER на следующих два вопроса. Если программа набрана Вами без ошибок, то после выполнения трансляции в текущем каталоге появится файл PRIMER11.OBJ. Это объектный файл (или модуль). Следующий этап процесса трансляции — это компоновка. Запустим программу LINK.EXE. В ответ появится строка: Object Modules [.OBJ]:. В ответ мы опять набираем имя нашей программы, т.е. PRIMER11 и нажимаем ENTER. Далее, как и в случае с MASM, на все вопросы отвечаем нажатием ENTER. В результате компоновки в текущем каталоге должна появиться программа PRIMER11.EXE - EXE-программа. Запуская ее, мы будем получать на экране строку: "Это Ваша первая программа на языке ассемблера". Конечная цель достигнута.

Трансляцию можно провести быстрее, если воспользоваться возможностями командной строки программ MASM.EXE и LINK.EXE. Если записать командную строку в виде MASM PRIMER11;, то трансляция будет произведена сразу без всяких вопросов ("точка с запятой" в конце необходима). Аналогично для компоновки можно набрать строку: LINK PRIMER11;. В строке можно набирать и другие команды, но об этом будет сказано несколько позднее.

Конечно, Вы знаете, что в операционной системе MS DOS, кроме программ типа EXE, существуют программы с расширением COM (COM-программы). Вы еще не готовы к тому, чтобы детально разобраться, в чем различие между ними, но кое-что Вы должны знать уже сейчас:

1. Длина COM-программ не превышает 64 Кб, тогда как для EXE-программ такого ограничения не существует.

2. Загрузка COM-программ происходит несколько быстрее, чем EXE-программ (впрочем, для быстрых компьютеров это практически незаметно).

3. Если Вы набрали в командной строке имя программы, скажем, AB, то вначале операционная система ищет файл AB.COM. Если он будет найден, то операционная система его запустит. В противном случае операционная система начинает искать файл AB.EXE. Помните об этом, когда создаете COM-программы. К слову будет сказано, BATCH-файлы ищутся операционной системой в третью очередь.

Некоторые EXE-программы могут быть преобразованы в COM-программы. Наша программа PRIMER11.EXE относится к их числу. Для этого воспользуемся утилитой EXE2BIN.EXE. Наберем следующую строку: EXE2BIN PRIMER11.EXE PRIMER11.COM. В результате появится программа PRIMER11.COM. Запустив ее, Вы получите тот же результат, что и для PRIMER11.EXE. Кстати, обратите внимание на длины этих двух программ. Другой способ получения COM-программ будет рассмотрен позднее. Если в Вашем распоряжении имеется LINK.EXE версии 5.0 и выше, то EXE2BIN.EXE не понадобится. При компоновке COM-программ следует просто набрать LINK PRIMER11 /T; - тогда из объектного файла сразу получится COM-файл (если это возможно). Замечу еще, что тот факт, что вдша программа правильно работает и в EXE, и в COM-виде (чего прак-

тически никогда не бывает), результат ее особой структуры. Я советую Вам в дальнейшем вернуться к ней и разобраться, в чем здесь дело.

Использование BATCH-файлов может значительно упростить процесс трансляции и отладки программ. Для тех, кто хорошо знаком с операционной системой MS DOS, приведем возможный и очень простой вариант такого BATCH-файла (Рис, 1.2.).

```
@ECHO OFF
:2
WD %1.ASM
ECHO НАЧИНАЕМ ТРАНСЛЯЦИЮ
MASM %1;
IF ERRORLEVEL 1 GOTO 1
ECHO ВЫПОЛНЯЕМ КОМПОНОВКУ
LINK %1.OBJ;
IF ERRORLEVEL 1 GOTO 3
ECHO ЗАПУСКАЕМ ПРОГРАММУ
PAUSE
@ECHO ON
%1
@ECHO OFF
ECHO ПРОГРАММА ВЫПОЛНЕНА, ПРОВЕРЬТЕ РЕЗУЛЬТАТ
PAUSE
GOTO 2
:1
ECHO ВО ВРЕМЯ ТРАНСЛЯЦИИ БЫЛИ ОБНАРУЖЕНЫ ОШИБКИ
PAUSE
ECHO ТРАНСЛИРУЕМ С ОБРАЗОВАНИЕМ ЛИСТИНГА
ECHO ДАЛЕЕ ЛИСТИНГ БУДЕТ ЗАГРУЖЕН В РЕДАКТОР
MASM %1,,;
WD %1.LST
PAUSE
GOTO 2
:3
ECHO ВО ВРЕМЯ КОМПОНОВКИ БЫЛИ ОБНАРУЖЕНЫ ОШИБКИ
PAUSE
GOTO 2
:END
```

Рис. 1.2. Вариант пакетного файла для трансляции программы с языка ассемблера.

Пусть имя пакетного файла будет GO.BAT, тогда, для того чтобы использовать его для трансляции программы PRIMER11.ASM, наберите следующую строку GO PRIMER11 и нажмите ENTER. Выйти из пакетного файла можно как обычно, т.е. посредством нажатия CTRL-C. А сейчас несколько пояснений. В командной строке для MASM указывается исходный файл, затем имя объектного файла, Далее имя файла листинга и, нако-

нец, имя файла перекрестных ссылок (.CRF). Параметры отделяются друг от друга запятыми. Например, строка MASM PRIMER11,P1,P1,PRI приведет к тому, что, кроме файла PRIMER11.ASM, на диске появятся также P1.OBJ - объектный файл, P1.LST - файл листинга, PRI.CRF - файл перекрестных ссылок. Если же вместо соответствующего параметра стоят две запятые, то это означает, что соответствующий файл должен иметь то же имя, что и исходная программа, но соответствующее расширение.

Приведем теперь текст еще одной программы, на которой Вы сможете отработать то, что здесь узнали. Обращаю Ваше внимание на то, что получившаяся программа не может быть преобразована к COM-виду.

```
;
TITLE   PRIMER13
;сегмент данных
DATSEG   SEGMENT
TEXT1 DB  'ПРИВЕТ, ПРИВЕТ!',10,13, '$'
DATSEG   ENDS
;сегмент стека
STSEG     SEGMENT STACK
          DB 60 DUP(?)
STSEG    ENDS
;сегмент кода
CODSEG   SEGMENT
ASSUME CS:CODSEG, DS:DATSEG, SS:STSEG, ES:CODSEG
BEGIN:
;следующие две строки излишни,
;т.к. у сегмента STSEG мы указали
;тип STACK
     MOV AX,STSEG
;SS должен указывать на сегмент стека
     MOV SS,AX
     MOV AX,DATSEG
;DS должен указывать на тот сегмент
; где находятся данные
     MOV DS,AX
;теперь вывод строки (см. Рис. 1.1.)
     LEA DX,TEXT1
     MOV AH,9
     INT 21H
;выходим в DOS
     MOV AH,4CH
     INT   21H
CODSEG   ENDS
     END BEGIN
```

Рис. 1.3. Пример еще одной программы, которая выводит на экран строку символов.

Программа всего лишь выводит на экран строку и заканчивает свою работу. В программе присутствуют три сегмента: кода, данных, стека. Замечу, что сегмент стека хоть и присутствует в программе, но, в общем, для порядка. Проделайте такой эксперимент: уберите сегмент стека из программы, а также все, что на него ссылается. Затем оттранслируйте программу, не обращая внимания на предупреждение, которое по этому поводу сделает Вам редактор связей LINK.EXE. Запустите программу и убедитесь, что она по-прежнему работает исправно. Суть здесь в том, что стек служит для временного хранения данных, а если Вы его в программе не используете, то присутствие или отсутствие его на работе программы никак сказаться не может.

В заключение хотелось бы поговорить об общей идеологии трансляции с языка ассемблера. Возникает законный вопрос: зачем нам нужны промежуточные (объектные) файлы? На первый взгляд это действительно несколько замедляет процесс разработки программы. Но это только на первый взгляд. Дело в том, что из объектных файлов можно создавать объектные библиотеки (не перепутайте с библиотеками объектов) процедур, которые в готовом виде можно использовать затем для разработки других программ. Конечно, библиотеку можно иметь и в текстовом варианте[5]. Но, во-первых, она будет занимать гораздо больше места, а, во-вторых, подсоединение объектной библиотеки лишь на последнем этапе трансляции как раз и ускоряет этот процесс. Ну и, наконец, поскольку структура объектного файла является стандартом не только для ассемблера, но и для многих трансляторов с языков высокого уровня, появляется удивительная возможность стыковки программ, написанных на разных языках. Становится возможным не только подсоединять процедуры, написанные на ассемблере к программам на таких языках, как Паскаль, Си, Фортран, Пролог и т.д., но и стыковать между собой, скажем, Паскаль и Си (см. Главу 15). На Рис. 1.4. представлена схема процесса трансляции программы, написанной на языке ассемблера.

Большинство трансляторов с языка ассемблера работают в два прохода[6]. При первом проходе транслятор составляет таблицу всех имен, встречающихся в программе, присваивая им соответствующее числовое значение (адрес). Во втором проходе происходит кодирование программы - замена команд на их числовые коды с учетом значений из таблицы. В результате получается объектный модуль, в котором наряду с кодами команд содержится информация о глобальных переменных, которая далее используется редактором связей.

И еще. Использование объектных библиотек удобно с точки зрения разработки программ. Однако одна и та же библиотека многократно дублируется в различных исполняемых модулях. Тем самым мы расходуем дополнительное место на жестком диске. Такие библиотеки можно назвать статическими. В более совершенных операционных системах (о новой операционной системе Windows, в которой возможно использование динамических библиотек, Вы узнаете в последующих главах) используют так называемые динамические библиотеки. Эти библиотеки подключаются только на стадии исполнения программы. Тем самым мы значительно экономим дисковое пространство, а также оперативную память.

---

[5] Текстовый модуль (библиотеку) можно подключить в нужное место программы при помощи директивы INCLUDE.

[6] Существуют одно- и многопроходные ассемблеры.

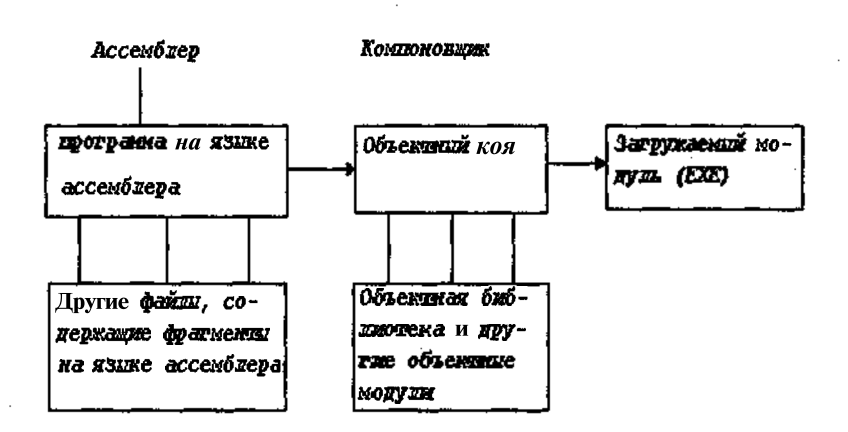


Рис. 1.4. Схема трансляции программы, написанной на языке ассемблера.

Отступление: об именах переменных, меток и другом.

В толстых и умных книжках Вы встретите утверждение о том, что имена, используемые в программах, должны быть "говорящими". Т.е., встретив переменную, Вы или кто-то другой сразу должен понять, для чего она используется. Один из подходов в создании таких говорящих имен называется "венгерской нотацией". В ней предполагается включать в имя переменной и ее тип. Сразу скажу, что я против использования подобных правил программистами. Изысканиями в этой области, на мой взгляд, занимаются люди, не слишком любящие программировать. Ничего не поделаешь, но правила часто придумывают люди, которые данным делом не занимаются (уже, еще или вообще). Хотите совет: занимайтесь придумыванием имен, только если Вы четко понимаете, зачем Вам это нужно. Длинные имена, кстати, часто мешают программировать. При написании длинного имени с большей вероятностью допускается ошибка. Тексты программ с венгерской нотацией вызывают у меня аллергию. Я совершенно не вижу смысла в использовании имени lpszClassName вместо простого Name. Впрочем, дело вкуса: "Кому нравится поп, а кому - попова дочь". Мне лично нравится программировать, а не выдумывать имена. В моих программах поэтому есть говорящие имена, но много и неговорящих. Зато есть комментарии, которые помогут Вам разобраться в алгоритме.

Глава 2. Адресное пространство, структура программ.

Нет, - ответила Маргарита,- более всего меня поражает, где все это помещается.

М.А. Булгаков. Мастер и Маргарита.

Кто может вместить, да вместит.

Евангелие от Матфея.

Программы для компьютера состоят из команд микропроцессора, представляющих собой машинные коды. Для выполнения программы она должна быть помещена в память компьютера. Специальный регистр микропроцессора содержит всегда адрес следующей исполняемой команды. В памяти компьютера содержатся и данные, которые могут использоваться исполняемой программой. Нет никакого различия между командами микропроцессора и данными. В определенных ситуациях команды программы могут рассматриваться как данные и, наоборот, данные становятся фрагментами программы. Таким образом, нет никакого препятствия к тому, чтобы программа в процессе выполнения изменяла саму себя. В языках высокого уровня введены препоны к самомодификации программ, при программировании же на ассемблере Вы свободны делать любые, в том числе и самомодифицирующиеся программы. "Дурной вкус!" - воскликнут приверженцы "высокого штиля" в программировании. Давайте будем терпимее - кому что нравится. Нельзя же все время слушать Баха.

Память ЭВМ состоит из однобайтовых ячеек. Каждой ячейке присваивается адрес - номер по порядку (от 0 и далее). Такой адрес мы будем называть физическим. Однако применительно к IBM PC принято представлять адрес в виде двух компонент. Связано это с тем, что регистры микропроцессора не могут вместить числа длиной большей двух байт (напоминаем читателю, что мы пока рассматриваем микропроцессор 8086, другие представители данного семейства будут рассмотрены позднее) - максимальное число FFFFH (шестнадцатеричная система!). Поэтому для формирования адреса используются два регистра (см. ниже). Используя же двухкомпонентное представление, можно адресовать память до одного мегабайта.

Двухкомпонентный (логический) адрес мы будем записывать в виде SA:OA, где SA - адрес сегмента, OA - смещение в этом сегменте. Обычный физический адрес ячейки памяти можно получить из двухкомпонентного адреса по формуле: SA*16+OA. Умножение на 16 равносильно сдвигу влево на четыре бита. Таким образом мы получаем возможность оперирования 20-битовыми адресами. Использование двухкомпонентного адреса с необходимостью приводит нас к разбиению памяти на сегменты (sic!). Размер сегмента не может превышать 64 Kb. Шестнадцатибайтовую величину принято называть параграфом. Легко видеть, что сегмент должен начинаться на гра-

нице параграфа. Отметим также, что если физический адрес у ячейки один, то двухкомпонентных адресов у нее может быть несколько, разумеется, все они будут равноправны. Например, двухкомпонентные адреса FF3AH:2367H и FF38H:2387H указывают на одну и ту же ячейку. Лишний раз подчеркнем, что сегментация памяти - следствие структуры микропроцессора, а именно размеров его адресных регистров. Начиная с микропроцессора 80386, 32-битные регистры позволяют адресовать память без использования сегментации. Фактически вся память в такой модели рассматривается как один большой сегмент - микропроцессор в этом случае непосредственно оперирует с физическими адресами. Но об этом речь еще впереди.

Используя служебные слова, в программе на ассемблере можно резервировать определенное количество байт, которые потом можно использовать для хранения каких-либо данных. Вот эти служебные слова: DB - резервировать байт, DW - резервировать слово (два байта), DD - резервировать двойное слово (четыре байта), DQ - резервировать восемь байт, DT - резервировать десять байт. Рассмотрим следующий фрагмент:

```
DB 78H        ;будет зарезервирован байт, которому присвоится
              ;значение 78H
DW 1234H      ;будет зарезервировано два байта, причем в
              ;младший
              ;будет помещено число 34H, а в старший 12H (!)
DQ 02f503     ;будет зарезервировано восемь байт, причем байты
              ;в памяти будут располагаться в следующем
              ;порядке:
              ;3,F5h,2,0 (старший байт оказался равным нулю!)
DB 12,34      ;будет зарезервировано два байта: младший 12,
              ;старший 34
```

Если в программе на языке ассемблера встречается символ или группа символов в кавычках, то транслятор поменяет их на байты, соответствующие их ASCII кодам. Например:

```
DB 'Ошибка вычисления' ;резервируется семнадцать байт, куда
                       ;помещаются байты, соответствующие
                       ;ASCII кодам записанной строки
                       ;см. также Рис.1.1.
```

Адресом любой из выше указанной цепочки считается адрес младшего байта. Предположим, в Вашей программе будет следующая строка: LAB DW 7589H. Тогда команда MOV AL,BYTE PTR LAB загрузит в однобайтный регистр AL микропроцессора число 89H. Как Вы уже, наверное, догадались LAB, есть не что иное, как метка [7].

[7] Точнее было бы назвать ее переменной. Метки в программе на языке ассемблера указывают на команды, переменные - на блоки данных. Однако в языке ассемблера, как и в машинном языке, нет различия между кодом и данными. В любой момент команды можно рассматривать как данные, а блок данных как фрагмент программы.

После трансляции всем меткам программы присваивается соответствующее числовое значение - смещение в сегменте. Служебные слова перед LAB (BYTE PTR) означают, что загружается именно байт. Для того чтобы загрузить старший байт указанного слова, следует выполнить команду MOV AL,BYTE PTR LAB+1. Для загрузки всего слова, скажем, в регистр DX, служит команда MOV DX,LAB.

Чтобы резервировать группу байт, удобно использовать служебное слово DUP. Ниже в программах Вы увидите, как это делается.

Работать с ячейками памяти можно по-разному. С первым способом адресации Вы уже познакомились - можно просто указать метку. Такая адресация называется прямой. К метке можно добавить какое-либо число. Другим способом адресации является косвенная адресация. Она использует регистр BX. Команда MOV AX,[BX] означает загрузить в регистр AX слово, адрес которого лежит в регистре BX. Адрес в регистр BX можно загрузить командой MOV BX,OFFSET LAB или LEA BX,LAB. К регистру BX можно добавлять смещение, например, MOV AX,[BX]+2 или просто MOV AX,[BX]2. Такая адресация называется адресацией по базе. Результирующий адрес получается сложением содержимого BX и смещения.

Следующий способ адресации называется прямой адресацией с индексированием. Здесь используют индексные регистры DI и SI. Например, следующие две команды

```
MOV DI,8
MOV AX,LAB[DI]
```

приведут к тому, что в регистр AX будет загружено слово, находящееся по адресу LAB+8. Последний способ адресации - это адресация по базе с индексированием. Приведем здесь просто команду без комментария: MOV CX,[BX][DI]+2.

Регистр BP очень похож на регистр BX, с тем лишь различием, что по умолчанию он адресует ячейки сегмента стека:

```
MOV AX, [BX] - загрузка из сегмента данных (DS),
MOV AX, [BP] - загрузка из сегмента стека  (SS).
```

Впрочем, сегмент можно указать явно, и тогда команда MOV AX,SS:[BX] - будет загружать слово из сегмента стека. Аналогично для BP - команда MOV AX,DS:[BP] загружает из сегмента данных (точнее, из сегмента, на который показывает регистр DS).

Описанные способы адресации памяти можно использовать не только в командах загрузки, но при выполнении различных арифметических операций. Например, команда ADD DI,LAB[SI] прибавляет содержимое ячейки, адрес которой получается сложением LAB и содержимого SI, к числу, которое находится в регистре DI. Результат сложения помещается в регистр DI.

Перейдем теперь непосредственно к структуре программы. Мы с Вами познакомились с таким понятием, как сегмент, и чем обусловлено введение этого понятия. Поскольку предполагается, что наши программы после трансляция будут загружаться в память, то и сама программа должна быть разбита на какое-то количество сегментов, что Вы и должны были заметить в предложенных Вам примерах

(см. Рис. 1.1 и Рис. 1.3). Минимальное количество сегментов - один. Вот как раз такие программы и могут быть преобразованы к СОМ-виду. Классическая структура программы состоит из трех сегментов: сегмент кода, сегмент данных, сегмент стека. Данная структура как раз соответствует программе на Рис. 1.3. Но мы не обязаны ограничивать себя только тремя сегментами - можно, например, сделать два сегмента данных, три сегмента кода и т.д. и т.п. Кстати, сейчас Вам должно быть понятно, почему длина СОМ-программы не может превышать 64К (максимальный размер сегмента).

Теперь настал черед поговорить о сегментных регистрах - в микропроцессорах 8088/8086 их всего четыре. В них хранятся сегментные адреса: CS - сегмент кода, DS - сегмент данных, SS - сегмент стека, ES - дополнительный сегмент. Чтобы не было недоразумений, хочу подчеркнуть, что это не означает, что в программе может быть только четыре сегмента. Например, вначале DS указывал на один сегмент данных, затем на другой - то же можно сказать и о других регистрах. На Рис. 2.1 представлена программа с двумя сегментами данных.

```
;-----------------------------------------------------------
TITLE  PRIMER21 .
;сегмент данных 1
D1SEG  SEGMENT
TEXT1  DB 'ПРОГРАММА НАЧИНАЕТ РАБОТАТЬ.$'
D1SEG  ENDS
;сегмент данных 2
D2SEG  SEGMENT
TEXT2  DB  'ПРОГРАММА ЗАКАНЧИВАЕТ СВОЮ РАБОТУ.$'
D2SEG ENDS
;сегмент стека
;резервируется 30 байт
;впрочем в данной программе стек не используется
STSEG SEGMENT STACK
      DB 60 DUP(0)          ;30 байтам присваивается значение 0
STSEG ENDS
CODSEG  SEGMENT
ASSUME CS:CODSEG, DS:D1SEG, SS:STSEG
BEGIN:
;устанавливаем сегмент стека
      MOV AX,STSEG
      MOV SS,AX
;DS направляем на первый сегмент данных
      MOV AX,D1SEG
      MOV DS,AX
      LEA DX,TEXT1   ;DS:DX теперь указывают на TEXT1
      MOV AH,9
      INT 21H
```

```
;ждем нажатие клавиши
     MOV AH,0
     INT 16H
;DS направляем на второй сегмент данных
     MOV AX,D2SEG
     MOV DS,AX
     LEA DX,TEXT2   ;DS:DX теперь указывают на TEXT2
     MOV AH,9
     INT 21H
;выходим в операционную систему
     MOV AH,4CH
     INT 21H
CODSEG  ENDS
     END BEGIN
```

Рис. 2.1. Пример программы с двумя сегментами данных.

А вот программа с двумя сегментами кода. Возможно, в ней Вам не все понятно, но это дело времени. Обращаю Ваше внимание на то, что переход между сегментами кода осуществляется командой длинного перехода. Фактически команда длинного перехода представляет собой одновременную замену содержимого сразу двух регистров CS и IP.

```
;сегмент данных
DATA SEGMENT
TEXT1 DB 'Я в сегменте 1',13,10,'$'
TEXT2 DB 'Я в сегменте 2',13,10,'$'
DATA ENDS
;сегмент стека
STSEG SEGMENT STACK
     DW 20 DUP(?)
STSEG ENDS
;первый сегмент кода
CODE1 SEGMENT
     ASSUME CS:CODE1, DS:DATA
BEGIN:
     MOV AX,DATA
     MOV DS,AX
     MOV AX,STSEG
     MOV SS,AX
     LEA DX,TEXT1
     MOV AH,9
     INT 21H
;переходим во второй сегмент
     JMP FAR PTR BEG_CODE2
```

```
_C,ODE1:
      MOV  AH,4CH
      INT  21H
CODE1 ENDS
;второй сегмент кода
CODE2 SEGMENT
      ASSUME CS:CODE2, DS:DATA
BEG_CODE2:
      LEA  DX,TEXT2
      MOV AH, 9
      INT  21H
;возвращаемся в первый сегмент
      JMP FAR PTR _CODE1
CODE2 ENDS
      END BEGIN
```

Рис. 2.2. Пример программы с двумя сегментами кода.

Выше было сформулировано необходимое условие того, чтобы программа могла быть преобразована к COM-формату. Назовем сейчас и другие условия. Вот они:

1. В командах программы не должны присутствовать имена сегментов (точнее, сегмента, т.к. для преобразования необходимо наличие лишь одного сегмента).

2. В конце программы после служебного слова END нужно указать метку начала программы. Дело в том, что COM-программа не может содержать какую-либо информацию для загрузчика, в том числе и точку входа в программу. Для EXE-программы точка входа может быть в любом месте программы. В принципе, для COM-программы метку можно не указывать вообще, но в этом случае придется учитывать сдвиг адресов (на 100H байт).

3. Используя директиву GROUP можно преобразовать к COM-формату программу, состоящую из нескольких сегментов (см. Приложение 3). Сегмент стека в такой программе, однако, должен отсутствовать.

Служебное слово ASSUME является директивой транслятора. Оно сообщает ему значение сегментных регистров. Для CS такая директива обязательна. Для регистров DS,SS и ES эти указания ассемблеру можно не делать. Иногда, однако, могут возникнуть некоторые затруднения. В частности, может появиться сообщение о фазовой ошибке (Phase error) в случае ссылки вперед: MOV AL,LL, где LL находится в старших адресах (точнее ниже). Ошибка при трансляции возникает из-за неопределенного значения регистра DS для ассемблера. При этом не важно, что программно сегментный регистр определен: MOV AX,DATA/MOV DS,AX.

При запуске COM-программ операционная система устанавливает вершину стека в старший адрес сегмента программы (подробнее о стеке см. Главу 3). Стек, таким образом, будет расти в сторону программного кода. Если пространства для стека не достаточно, то вершину следует установить в старшие адреса памяти.

Для оперативного! управления памятью DOS разбивает ее на блоки, каждый из которых имеет заголовок. Такое деление является недокументированным, т.е. разра-

ботчики не гарантируют, что в последующих версиях операционной системы они не откажутся от такого подхода. Однако MCB (Memory Control Block - контрольный блок памяти) сохранился вплоть до седьмой версии MS DOS, и, судя по всему, фирма Microsoft не собирается с ним расставаться.

Итак, вся память делится на блоки, а каждый блок имеет заголовок длиной 16 байт. Ниже (Рис. 2.3) приведена структура такого заголовка.

Смещение	Длина поля	Содержание
0	1	M - если блок не последний Z - если блок последний.
1	2	сегментный адрес владельца, 0 если блок памяти не занят.
3	2	длина блока в параграфах (SIZE); если S - сегментный адрес данного заголовка, то заголовок следующего блока будет находиться по адресу S+SIZE+1.
5	3	Резерв
8	11	зарезервировано; начиная с пятой версии MS DOS первые восемь байт используются для имени владельца.

Рис. 2.3. Структура заголовка блока памяти.

Благодаря использованию блочной структуры имеется возможность осуществлять некоторое подобие многозадачного режима. Программа может быть запущена, а затем остаться в памяти. Причем программы, которые будут запускаться после нее, не смогут повредить ее код, если будут пользоваться стандартными DOS'овскими процедурами. Просто свободной памяти для них станет меньше. Такой подход используется в частности для создания резидентных драйверов, которые, постоянно находясь в памяти, осуществляют управление какими-либо внешними устройствами. С резидентными программами мы с Вами еще не раз встретимся. В частности, этому вопросу будут посвящены несколько глав. Кстати, большая часть вирусов имеет как раз резидентный характер.

Ни в коем случае не следует портить заголовок блоков - это может привести только к катастрофическим результатам, т.к. операционная система постоянно проверяет целостность всей цепочки блоков.

Рассказ наш будет неполным, если не упомянуть еще об одной структуре - векторе. Вектором называют адрес некоторой процедуры, расположенной в памяти. Причем в двух старших байтах располагается адрес сегмента, а в двух младших - смещение. Предположим, при просмотре памяти обнаружена следующая цепочка байт, которая, как Вы полагаете, является вектором: F26778BOH. Адрес процедуры будет тогда следующий: B078H:67F2H (объясните почему). Начало оперативной памяти отводят как раз под такие векторы, которые операционной системой во время ее загрузки устанавливаются на свои процедуры. Векторы принято нумеровать, причем если номер вектора N, то он располагается по адресу 0:4*N. Команда INT 21H - есть не что

иное, как вызов процедуры операционной системы, на которую указывает вектор под номером 21Н (он расположен по адресу 0:84Н=21Н*4). Аналогично INT 16Н - вызов процедуры, на которую направлен вектор с номером 16Н. Такие процедуры мы в дальнейшем будем называть прерываниями. Умение грамотно работать с векторами прерываний особенно необходимо при написании резидентных программ (см. Главы 11,12 и 17). В заключение приведем карту памяти IBM PC (Рис. 2.4.).

Карта, приведенная на Рис. 2.4, является не слишком подробной. В литературе Вы сможете найти более подробную картину. Однако на данном этапе нам вполне достаточно такой карты. В процессе движения вперед Вы более подробно узнаете те области, которые приведены на рисунке. Используя утилиту -UTIL.COM из антивирусного пакета Е. Касперского (старого), Вы сможете и самостоятельно изучать структуру памяти операционной системы MS DOS, что, несомненно, доставит Вам большее удовольствие, чем рассматривание данного рисунка.

0000Н:0000Н	Начало таблицы векторов
0040Н:0000Н	Данные BIOS
0050Н:ОООOН	Данные DOS
ХХХХН:0000Н	Область памяти, куда загружаются файлы IO.SYS и MSDOS.SYS И где операционная система хранит свои данные. Здесь же помещаются загружаемые данные.
ХХХХН:0000Н	Здесь помещается резидентная часть COMMAND.COM
ХХХХН:0000Н	Область, где обычно располагаются резидентные программы.
ХХХХН:0000Н	Область памяти, куда помещается запускаемая программа.
ХХХХН:0000Н	Область памяти, куда помещается транзитная часть COMMAND.COM. Может затираться исполняемой программой. По выходу в MS DOS восстанавливается с диска.
А000Н:0000Н	Начало видеопамяти для VGA адаптера. В текстовом режиме используется только 32 К., начиная с адреса 0В800Н.
С000Н:0000Н	Начало ПЗУ.

Рис. 2.4. Карта памяти операционной системы MS DOS. ХХХХН означает переменный адрес.

Прокомментируем Рис. 2.4. Во-первых, область до начала видеопамяти называется базовой. Эту память в основном и использует операционная система MS DOS, здесь помещаются исполняемые программы и данные, ими используемые. Адресное пространство, начиная с А000:0000 и до конца ПЗУ, называют UMB - Upper Memory Block, т.е. верхний блок памяти. Этот блок, однако, не полностью используется ПЗУ и видеопамятью. Незанятые блоки используются для эмуляции так называемой дополнительной памяти (expanded). Эмуляция дополнительной памяти на машинах класса ЕХТ

производилась с помощью специальной аппаратуры и программного обеспечения. На машинах класса AT дополнительная память получается с помощью специальных драйверов из расширенной памяти (см. ниже). Кроме того, на машинах AT 80386 появилась возможность использовать UMB для переноса части ПЗУ в ОЗУ. Для этого также необходимо наличие расширенной памяти. Расширенная память - эта память с адресами свыше 1 Мб. Если известно, что на Вашем компьютере имеется 8 Мб памяти, то это означает, что часть ее входит в базовую память - 640 Кб, остальная память - расширенная (т.е. приблизительно 7.3 Мб). Доступ к расширенной памяти можно получить через прерывания 15H (см. Глава 5) или посредством перехода в защищенный режим (см. Главу 20). Подробно о расширенной и дополнительной памяти см. Главу 22. Подчеркнем лишний раз, что деление памяти на базовую и расширенную память есть прерогатива операционной системы MS DOS и является следствием ее несовершенности. Операционная система Windows лишена этого недостатка (см. главы 24,25).

Вопросы структуры программы, затронутые в данной главе, найдут свое дальнейшее развитие в Главе 13.

В заключение мне хотелось бы представить Вам (возможно, несколько забегая вперед) забавную, на мой взгляд, программу (Рис. 2.5). Она демонстрирует следующие сделанные мною утверждения:

1. Нет никаких различий между данными и кодом программы.

2. Из первого утверждения следует, что нет различия между меткой и переменной, поскольку обе они указывают на некоторую область памяти. Транслятор, правда, пытается слабо противиться такому "кощунству", но мы легко обманываем его ("Ах, обмануть меня не трудно - я сам обманываться рад!").

```
DATA  SEGMENT
      ASSUME CS:DATA ;строка необходима, чтобы использовать
;метку в данном сегменте - обман транслятора
;эти команды будут перемещены в сегмент кода и там выполнены
L1:   ;метка и одновременно переменная
;вывод буквы A на экран
      MOV DL,65 /два байта
      MOV AH,2  ;два байта
      INT 21H   /два байта
DATA  ENDS
STAC  SEGMENT STACK
      DB 50 DUP(?)
STAC  ENDS
CODE  SEGMENT
      ASSUME CS:CODE, SS:STAC, DS:DATA
BEGIN:
      MOV AX,DATA
      MOV DS,AX
      LEA SI,L1
      LEA DI,L2
```

```
;перемещаем данные из сегмента данных в сегмент кода
    MOV  AX,WORD PTR  [SI]
    MOV WORD PTR CS:[DI],AX
    MOV AX,WORD PTR [SI]+2
    MOV WORD PTR CS:[DI]+2,AX
    MOV AX,WORD PTR [SI]+4
    MOV WORD PTR CS:[DI]+4,AX
;необходимость такого перехода будет осмыслена Вами
;несколько позднее
    JMP L2
;резервная область, куда будут перемещены команды
;из сегмента команд
L2 :   ;метка и одновременно переменная
    NOP
    NOP
    NO.P
    NOP
    NOP
    NOP
    MOV AH,4CH
    INT 21H
CODE ENDS
    END BEGIN
```

*Рис. 2.5. Демонстрирует **возможность** использования команд в качестве данных.*

Если программа на Рис. 2.5 покажется Вам непонятной - вернитесь к ней после главы 4.

Глава 3. Первые программы.

- Лед тронулся, господа присяжные заседатели.

Остап Бендер.

В данной главе мы приведем несколько программ и постараемся, как можно полнее объяснить их. Наша книга содержит обширный справочный материал - по функциям DOS и BIOS, по прерываниям, портам ввода-вывода, командам микропроцессора и др. Разбирая каждую из программ, не забывайте справляться в справочнике. Через некоторое время Вы почувствуете, что, пользуясь только справочником, способны самостоятельно писать программы. Произойдет чудо, и Вас уже никто не сможет оттащить от программирования.

Прежде чем разбирать программы, однако, взгляните на Рис. 3.1, где представлена простая схема функционирования микропроцессора Intel 8086[8]. Как видим из рисунка, микропроцессор поделен на две части: операционное устройство (ОУ) и шинный интерфейс (ШИ). Роль ОУ заключается в выполнении команд, в то время как ШИ подготавливает команды и данные к выполнению. Данная схема является программной и ни в коей мере не отражает детали технического функционирования процессора.

Операционное устройство содержит арифметико-логическое устройство (АЛУ), устройство управления (УУ) и десять регистров. Эти устройства обеспечивают выполнение команд, арифметические вычисления и логические операции. Указатель или регистр команд содержит относительный адрес той команды (относительно сегмента, на который указывает регистр CS), чья очередь сейчас выполняться. Регистр флагов содержит биты, значения которых определяют состояние микропроцессора или результат предыдущей арифметической команды. Значения этих битов, в частности, используются микропроцессором при выполнении команд условных переходов. Ниже регистр флагов будет рассмотрен более подробно. Следует обратить внимание на набор регистров. Эти регистры называют рабочими. В них можно хранить данные, обмениваться этими данными с памятью. Над данными, находящимися в этих регистрах, можно производить арифметические и логические действия. Все регистры двухбайтные. Однако регистры AX,BX,DX,CX обладают дополнительным свойством - можно оперировать однобайтовыми компонентами этих регистров. Например, наряду с регистром AX можно оперировать регистрами AL (младший байт) и AH (старший байт). То же можно сказать и о трех других регистрах.

Три элемента шинного интерфейса - блок управления шиной, очередь команд и сегментные регистры осуществляют следующие функции: во-первых, ШИ управляет передачей данных на операционное устройство, в память и на внешние устройства. Во-вторых, четыре сегментных регистра управляют адресацией памяти (см. предыдущую главу). Третья функция - управление очередью команд (буфером команд). Задача заключается в том, чтобы всегда существовала непустая очередь команд, готовых для выполнения. Кстати, именно существование очереди команд нужно учитывать, если Вы задумали написать самомодифицирующуюся программу (см. главы 4, 18).

---

[8] Набор регистров и их разрядность изменилась, начиная с процессора 80386 (см. Главу 20).

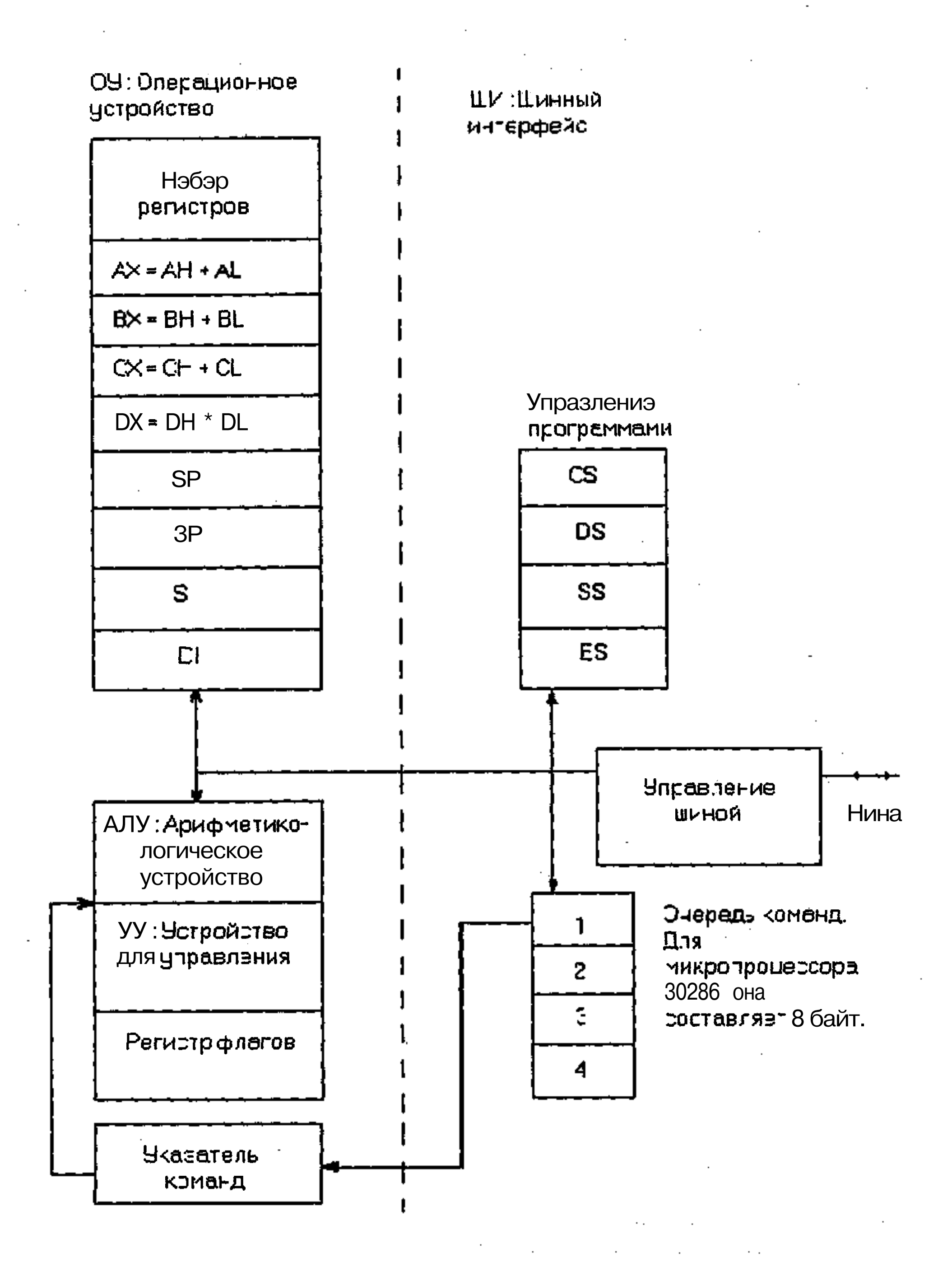


Рис. 3.1. Простая функциональная схема микропроцессора 8086/8088.

Итак, приступаем. Первая программа (Рис. 3.1) позволяет вводить строку символов, а затем выводит эту строку на экран. Для полного уяснения принципа работы программы Вам необходимо разобраться в работе функций 09Н и ОАН прерывания 21Н.

```
;———сегмент данных———
DSEG SEGMENT
;ниже представлена структура строкового буфера,
;с которым работает функция ОАН
MAX    DB 255             ;максимальная длина строки
LEN    DB ?               ;длин после ввода строки
STROKA DB 255 DUP(?)      ;буфер для строки
       DB ?               ;резервный байт
ENT    DB 13,10,'$'       ;перевод строки
DSEG ENDS
;сегмент стека
STSEG SEGMENT STACK
      DB 60 DUP(0)
STSEG ENDS
CODSEG SEGMENT
ASSUME CS:CODSEG, DS:DSEG, SS:STSEG
BEGIN:
;устанавливаем сегмент стека
      MOV AX,STSEG
      MOV SS,AX
;DS направляем на сегмент данных
      MOV AX,DSEG
      MOV DS,AX
;вводим строку
      MOV AH,0AH
      LEA DX,MAX    /указываем на буфер
      INT 21H
;готовим введенную строку для вывода по прерыванию 09H
      MOV BL,LEN
      MOV BH,0
;знак '$' является признаком конца строки
/для функции 09H
      MOV [BX+STROKA+1],'$'
;переведем строку
      LEA DX,ENT      ;указываем на строку
      MOV AH,09H
      INT 21H
/теперь печатаем введенную строку
      LEA DX,STROKA  /указываем на строку
      INT 21H
```

```
;функция 4CH прерывания 21H осуществляет корректный
;выход в операционную систему
     MOV AH,4CH
     INT 21H
CODSEG   ENDS
     END BEGIN
```

Рис. 3.1. Ввод и вывод строки посредством функций 09H и 0AH прерывания 21H.

Прокомментируем программу. Во-первых, рассмотрим команду MOV [BX+STROKA+1],'$'- засылка в конец строки символа $. Откуда ассемблер знает, что строка находится в сегменте данных? Дело в том, что если явно не указан сегментный регистр, то всегда предполагается, что данные находятся в том сегменте, на который указывает регистр DS. Более явно эту команду можно записать так — MOV DS:[BX+STROKA+1],'$'. Если же мы хотим послать данные в сегмент, на который указывает другой сегментный регистр, то его надо указать явно. Например — MOV ES:[BX+5],65. Теперь насчет того, что означает BX+STROKA+1. Берется адрес, соответствующий метке STROKA, и к нему прибавляется содержимое регистра BX, а затем 1. В результате мы как раз получаем адрес последнего символа введенной строки + 1. Аналогично можно сказать и о команде LEA DX,STROKA. В DX загружается адрес, соответствующий метке STROKA, относительно сегмента, адресуемого регистром DS; явная команда будет иметь вид LEA DX,DS:STROKA.

Во-вторых, почему мы только один раз засылаем в регистр AH номер функции - 09H? Оказывается, что при выполнении прерывания 21H значение всех регистров не меняется, за исключением тех случаев, когда в регистры возвращаются какие-то данные.

В-третьих, что означают байты 13,10? Это магическое сочетание для многих функций вывода на экран, и принтер имеет тот смысл, что следует перейти к началу следующей строки.

Возможно, не все ясно из моего комментария, но прервемся на этом и перейдем к следующей программе. Вы знаете, что некоторые программы могут возвращать в DOS коды ошибок. Эти коды можно определить в BATCH-файле при помощи конструкции IF ERRORLEVEL NUMBER. Возвратить код ошибки (код выхода) можно с помощью функции 4CH прерывания 21H (см. Рис.3.1), если в регистр AL предварительно поместить этот код. Программа будет ожидать нажатия клавиши, и если нажата цифровая клавиша, то будет происходить выход в DOS с возвратом кода (естественно, из промежутка от 0 до 9). Реально код возврата может иметь значение в пределах одного байта, т.е. до 255.

```
CODSEG   SEGMENT
ASSUME  CS:CODSEG
ORG 100H
BEGIN:
;--вызываем функцию 0 прерывания 16H--
```

```
;--ожидаем нажатие клавиши--
     MOV AH,0
     INT 16H
;--в AL возвращается код ASCII--
;--проверяем нажата ли цифровая клавиша--
;--если нет то повторяем ввод--
     CMP AL,48    ;сравнить AL и 48
     JB  BEGIN    ;перейти если ниже
     CMP AL,58    ;сравнить AL и 58
     JA  BEGIN    ;перейти если выше
;--из кода ASCII получаем цифру--
     SUB AL,48    /вычесть из AL 48
;--выходим в DOS, AL содержит код выхода--
     MOV AH,4CH
     INT 21H
CODSEG  ENDS
     END BEGIN
```

Рис. 3.2. Выход в DOS с передачей кода выхода.

Как Вы уже, несомненно, догадались, данная программа может быть преобразована к COM-виду. Несмотря на простоту, она может быть очень полезной при написании BATCH-файлов. На базе такой программы можно строить простейшие меню. Надеюсь, что текст программы достаточно прокомментирован. Мне же хочется обратить Ваше внимание на директиву, которая появилась, начиная с самых первых программ. Это ORG 100H. Она означает, что адреса всех команд, стоящих после этой директивы, должны вычисляться со смещением 100H. Для EXE-программы такая директива излишняя. Для COM-программы она необходима. Дело здесь вот в чем. При загрузке COM-программы в память она размещается не с начала сегмента, а со смещением 100H байт. В начале сегмента располагается некоторый блок служебной информации (PSP - Program Segment Prefix, т.е. префикс программного сегмента). PSP есть и у EXE-программ, но здесь при загрузке происходит дополнительная настройка адресов, поэтому проблем не возникает, а PSP имеет свой отдельный сегмент длиной 256 байт. Подробнее о PSP мы будем говорить позднее.

Следующая программа несколько сложнее предыдущих. Она ждет нажатия клавиши (имеющей код ASCII) и выдает ее символ. Если код расширенный, то выдается звуковой сигнал. Выйти из программы можно по нажатию CTRL-BREAK.

```
CODSEG  SEGMENT
ASSUME  CS:CODSEG
     ORG 100H
BEGIN:
     JMP BEGIN1
ENT DB 13,10,'$'        /данные для перевода строки
```

```
BEGIN1:
;--ожидаем нажатие клавиши--
     mov AH, 0
     INT 16H
;--проверяем: не расширенный ли код
     CMP AL,0
     JNZ DISP
;--если код расширенный то подаем звуковой сигнал--
     MOV AH,02
;--функция 2 прерывания 21H выводит символ, но
;--код 7 трактуется ей как звуковой сигнал--
     MOV DL,7
     INT 21H
;--возвращаемся к ожиданию нажатия клавиши--
     JMP BEGIN1
DISP:
;--символ выводим с помощью функции ОАН прерывания 10H--
     MOV AH,0AH
     MOV CX, 1        ;выводим один символ
     MOV BH,0         ;страница 0
     INT ЮH           ;вызов прерывания
;--теперь переводим строку и возвращаемся к началу--
     MOV AH,09
     MOV DX,OFFSET ENT
     INT 21H
     JMP BEGIN1
;--команда выхода в ДОС здесь совсем не нужна--
;--т.к. выход осуществляется по CTRL-BREAK--
CODSEG  ENDS
     END BEGIN
```

Рис. 3.3. Программа вывода символов нажатых клавиш.

Прежде всего, возникает вопрос: почему в программе используется два способа вывода символов? Дело в том функция 2 прерывания 21H выводит лишь обычный алфавит. Коды же меньшие 32 воспринимаются ей как команды: 13 - перевод строки, 7 - звуковой сигнал, 8 - сигнал табуляции и т.п. Функция ОАН прерывания ЮН лишена этого недостатка. В частности, Вы можете получить символ, соответствующий коду 7, нажав клавиши CTRL-G. Кстати, если Вы нажмете CTRL-C, то вместо выхода из программы получите символ, соответствующий данному сочетанию.

Чтобы перейти к следующей программе, введем несколько новых понятий. Прежде всего это стековая память. Если Вы занимались программированием, то с этим понятием Вы уже встречались. Стек - это организация памяти по принципу: первым пришел - последним ушел. Элементом стека является слово. В любой момент доступно

только слово, находящееся в вершине стека. На это слово указывает регистр SP (Stack Pointer - указатель стека). Сам же стек располагается в сегменте, на который в данный момент указывает сегментный регистр SS. Существуют две основные команды работы со стеком: PUSH - положить в стек, POP - вынуть из стека. При выполнении команды PUSH (например, PUSH AX, PUSH DX и т.п.) содержимое регистра SP уменьшается на 2, а в то слово, на которое теперь указывает этот регистр, помещается содержимое соответствующего регистра (в нашем примере AX или DX). При выполнении команды POP (например, POP DS, POP BX и т.п.), содержимое SP увеличивается на 2, а в соответствующий регистр (в нашем примере DS или BX) кладется то слово, на которое указывал до этого SP.

Кроме отмеченных команд PUSH и POP, со стеком также работают следующие команды: CALL - вызов процедуры, RET - возврат из процедуры, INT - вызов прерывания, IRET - возврат из прерывания. Рассмотрим, например, как работает пара CALL и RET. При выполнении команды CALL, например, CALL NET, где NET - метка перехода либо имя процедуры (для языка ассемблера это практически одно и то же), в стек кладется адрес возврата. При выполнении же команды RET этот адрес возврата из стека извлекается, и микропроцессор переходит к выполнению команды, следующей за командой CALL. Естественно, у вдумчивого читателя может возникнуть вопрос: "Какой адрес возврата - четырехбайтовый или двухбайтовый?" Обещаю, что мы вернемся к обсуждению данной проблемы в самое ближайшее время. Пока же будем считать, что переходы происходят в пределах одного сегмента и, следовательно, адрес возврата является двухбайтной величиной, т.е. словом.

Перейдем к рассмотрению следующего примера. Эта программа позволяет ввести строку символов, а затем выводит ее в обратном порядке.

```
;-- Сегмент данных —
DSEG        SEGMENT
;--структура для ввода строки (см. описание функции OAH)
MAX          DB  255
LEN         DB  0
STROKA      DB  256  DUP(?)
;--перевод строки
ENT     DB 13,10,'$'
DSEG        ENDS
;--Сегмент стека--
STSEG       SEGMENT  STACK
            DB  30  DUP(?)
TOP     DB  ?
STSEG    ENDS
;--Сегмент кода--
CODSEG    SEGMENT
      ASSUME   CS:CODSEG, DS:DSEG, SS:STSEG, ES:CODSEG
BEGIN:
      JMP  BEG
```

```
;--Процедура обращения строки--
CONVERT PROC
;--подготавливаем регистры--
     LEA DI,STROKA
     MOV BL,LEN
     MOV BH,0
     ADD BX,DI
     DEC BX          ;теперь BX показывает точно на конец строки
CICL:
     CMP DI,BX       ;если DI=>BX то заканчиваем
     JNB KONEC
;--производим обмен символов--
     MOV AL,[DI]
     MOV AH,[BX]
     MOV [DI],AH
     MOV [BX],AL
;--переходим к следующей паре символов--
     INC DI
     DEC BX
     JMP CICL
KONEC:
     RET             ;выходим в основную программу
CONVERT ENDP
;--Процедура вывода строки символов--
DISP_STR PROC
;--вначале переходим к следующей строке--
     MOV DX,OFFSET ENT
     MOV AH,09H
     INT 21H
;--подготавливаем регистры--
     MOV DI,OFFSET STROKA
     MOV CL,LEN
     MOV AH,02H         /функция прерывания 21H для вывода символа
PROD:
     MOV DL,[DI]
     INT 21H            /вывод символа из DS:[DI]
     INC DI             ;следующий символ
     DEC CL             /уменьшаем счетчик
     JNZ PROD           /если не 0 продолжаем вывод
     RET                ;выход в основную программу
DISP_STR ENDP
BEG:
;--устанавливаем сегментные регистры--
     MOV AX,DSEG
```

```
      MOV  DS,AX
      MOV  AX,STSEG
      MOV  SS,AX
      MOV  SP,OFFSET SS:TOP
;--осуществляем ввод  строки
      MOV  DX,OFFSET MAX   ;указываем на  структуру  для
;ввода  строки
      MOV  AH,0AH           ;функция ввода  строки
      INT  21H
      CMP  LEN,0            ;если строка  пустая,  то  выход
      JZ   EXIT
      CALL  CONVERT
      CALL  DISP_STR
EXIT:
      MOV  AH,4CH
      INT  21H
CODSEG   ENDS
      END  BEGIN
```

Рис. 3.4. Вывод строки в обратном порядке.

Комментируя программу на Рис.3.4, прежде всего хочу заметить, что, разумеется, если речь идет просто о выводе строки в обратном порядке, то нет нужды менять в строке порядок символов. С помощью той же функции 02H прерывания 21H можно вывести строку, взяв символы в обратном порядке. Вам предлагается реализовать именно этот подход для поставленной задачи.

В нашей программе впервые появилась такая структурная единица как процедура. Не надо, однако, смотреть на нее как, скажем, на процедуры в Паскале или функции в Си. Фактически это лишь другой способ указания метки перехода. Программа работала бы точно так же, если бы мы просто указали две метки: CONVERT и DIS_STR (разумеется, CONVERT ENDP и DIS_STR ENDP следовало бы при этом убрать). Правда, здесь не все так уж просто, и с дальнейшим развитием данного вопроса Вы познакомитесь в следующей главе. По поводу программы на Рис. 3.4 замечу также, что в конце вместо END BEGIN можно было бы поставить END BEG. Тогда не понадобилась бы команда JMP BEG. Впрочем, о подобных тонкостях речь еще впереди.

Две следующие программы иллюстрируют свойства стека, а также регистра BP (Рис. 3.5 - 3.6).

```
;первая  программа,  иллюстрирующая  свойства  стека
CODSEG    SEGMENT
ASSUME    CS:CODSEG
          ORG  100H
BEGIN:
      MOV AX,65
      PUSH  AX
```

```
/в стеке слово 0041H (65)
     MOV BP,SP
L1:
     CMP WORD PTR [BP],65
;BP как раз указывает на вершину стека, где лежит слово
;если бы это было не так, то программа никогда не закончила
;бы свою работу
     JNZ L1
     POP BX
;теперь слово 0041H находится в BX, проверим это
L2:
     CMP AX,BX
     JNZ L2
     LEA AX,EXIT
     PUSH AX
;в стек поместили смещение EXIT,
;поэтому RET работает как JMP EXIT
     RET
     JMP  BEGIN
EXIT:
     MOV AH,4CH
     INT 21H
CODSEG  ENDS
     END BEGIN
```

Рис. 3.5. Иллюстрация свойств стека (1).

```
;вторая программа, иллюстрирующая свойства стека
CODE  SEGMENT
     ORG 100H
     ASSUME CS:CODE
BEGIN:
     MOV  AH, 2
     MOV  DL,65
;--------------------
     PUSH AX
;следующие три команды работают как PUSH DX
     SUB  SP,2
     MOV  BP,SP
     MOV   [BP],DX
;--изменим содержимое регистров--
     MOV  DH,78
     MOV  AH, 56
```

```
;--------------------
        POP     DX
;следующие три команды работают как POP AX
        MOV     BP,SP
        MOV     AX, [BP]
        ADD     SP,2
;--вывод символа на экран--
        INT     21H
;--передать управление операционной системе--
        RET
CODE    ENDS
        END BEGIN
```

Рис. 3.6. Иллюстрация свойств стека (2).

Советую подробно разобраться в этих программах. Если Вы поняли, как они работают, то считайте, что поняли, как работает стек. Напомню также, что регистр BP для стека работает так же, как BX для сегмента данных. В языках высокого уровня, таких, как Си и Паскаль, при организации процедур (функций) в стеке располагаются локальные переменные, и доступ к ним как раз осуществляется посредством регистра BP (см. главы 11, 15).

Состояние регистра флагов влияет на выполнение команд микропроцессором. Сейчас мы рассмотрим структуру регистра флагов. Регистр флагов (Рис. 3.7) состоит из 16 бит[9].

15	14	13	12	11	10	9	8	7	€	5	4	3	2	1	O
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Рис. 3.7. Регистр, флагов микропроцессоров 8086/8088.

Из 16 бит семь зарезервированы, остальные выполняют различные функции, которые мы сейчас разберем.

CF - флаг переноса. Равен 1, если произошел перенос при сложении или заем при вычитании, в противном случае он равен нулю. При выполнении операции сдвига CF содержит бит, который вышел за границу ячейки или регистра. CF также служит индикатором результата умножения.

PF - флаг четности. Равен 1, если в результате операции получилось число с четным числом единиц, и 0 - в противном случае.

AF - вспомогательный флаг переноса. Аналогичен флагу CF, но контролирует заем или перенос третьего бита.

---

[9] Регистр флагов, начиная с процессора 80386 стал 32-битным (см. главу 20).

ZF - флаг нуля. Равен 1, если в результате операции получен нуль, и 0 - в противоположном случае.

SF - флаг знака. Дублирует значение старшего бита результата операции. Используется при работе с числами со знаком.

TF - флаг трассировки. Если этот бит равен 1, то после выполнения каждой операции микропроцессор обращается к специальной процедуре (прерыванию). Используется при отладке программы.

IF - флаг прерывания. Если данный флаг сброшен в 0, то микропроцессор не реагирует ни на какие внешние сигналы (сигналы прерывания). Исключение составляет немаскируемое прерывание (NMI). По линии NMI микропроцессор получает сообщения о таких критических ситуациях, как отключение питания и ошибка памяти.

DF - флаг направления. Используется строковыми (цепочечными) командами. Если он сброшен, цепочка обрабатывается с первого элемента, имеющего наименьший адрес. В противном случае цепочка обрабатывается от наибольшего адреса к наименьшему.

OF - флаг переполнения. Флаг равен 1, если результат сложения двух чисел с одинаковым знаком или результат вычитания двух чисел с противоположными знаками выйдет за пределы допустимого диапазона. Флаг обращается в 1, если старший бит операнда изменился в результате операции арифметического сдвига. OF=0, если частное от деления двух чисел переполняет результирующий регистр.

Выполнив команды PUSHF и POP AX (см. следующую главу), вы сможете проверить любой из вышеперечисленных флагов, даже если он никак не воздействует на команды условных переходов (см. следующую главу). Аналогично (PUSH AX/POPF) можно изменить любой флаг по своему усмотрению. Кстати, раз мы здесь говорим о регистре флагов, следует упомянуть один подход, который позволяет различать микропроцессоры Intel. Одна модификация микропроцессора может отличаться от предыдущей версии тем, что в ней появляется новый флаг. Это значит, что мы можем его изменить, а следовательно, отличить один микропроцессор от другого (например, 80386 от 80486) (см. главу 23).

Глава 4. Обзор команд микропроцессора 8088/8086.

МЕФИСТОФЕЛЬ

О, все пойдет па лад:
В редукцию лишь надо вникнуть
К классификации привыкнуть.

Фауст.
Гете.

Наберитесь терпения: данная глава будет посвящена обзору команд микропроцессора 8088/8086. Для тех, кто легко ориентируется в командах, приведенных в Приложении 1, эту главу можно опустить. Обзор будет достаточно кратким, т.к. о свойствах части команд Вы узнаете из приведенных далее программ. Сведения о командах, не нашедших отражения в главах, можно найти в Приложении 1. Впрочем, в данной главе Вы сможете также найти полезные фрагменты и программы.

Еще раз напомню, что речь идет о множестве команд микропроцессора 8088/8086. О микропроцессорах 80386 и выше с 32-битной арифметикой Вы прочтете в главе 20. Однако все микропроцессоры указанного семейства работают с этим набором команд. Более того, в большинстве случаев 32-битную арифметику нет никакой нужды использовать, т.к. для большинства переменных целого типа 16 бит вполне достаточно.

Программирование, дорогой читатель, сродни шахматам. Чтобы получать удовольствие от игры необходимо изучить не только фигуры и то, как они ходят, но и некоторые азы шахматной теории, а также получить определенные практические навыки. С тех пор как я увлекся программированием, я забросил шахматы.

I. Команды пересылки.

В главе 2 мы уже познакомились с командами пересылки в память из регистров микропроцессора и обратно. Можно обмениваться данными между регистрами: MOV AX,BX; MOV CL,DH и т.п. В регистры или память можно непосредственно загружать числа (байты или слова): MOV CX,2000; MOV M1,80. Относительно последней команды надо сказать следующее: Вы должны четко представлять, куда засылается число 80 - в слово или в байт. Это зависит от того, как определена метка M1 - DB или DW (говорят об атрибуте ячейки памяти, на которую указывает Ml). Если это слово, то тогда старший байт будет автоматически обнуляться. А если Вам нужно переслать число так, чтобы не затронуть старший байт, тогда пишите MOV BYTE PTR M1,80, если хотите переслать это число в старший байт, воспользуйтесь командой MOV BYTE PTR M1+1,80.

В наших программах Вы встречали команду типа MOV AX,DSEG - DSEG - это метка начала сегмента. Здесь ассемблер знает, о чем идет речь, и в AX загружается сегментный адрес, а не то, что лежит по этому адресу. Получить сегментный адрес можно и другим

способом. Если метка THERE находится в некотором сегменте, то, для того чтобы получить сегментный адрес, достаточно выполнить команду MOV AX,SEG THERE.

Для загрузки в регистр смещения какой-либо ячейки (адреса в сегменте) используют команды LEA AX,THERE или MOV AX,OFFSET THERE. В программах я намеренно использую и ту, и другую команды.

Очень удобна команда LES. Например, LES DX,KONEC заменяет сразу две команды:

```
MOV  DX,KONEC
MOV  ES,KONEC+2
```

Аналогично ведет себя команда LDS, но по отношению к регистру DS. Заметим, что в [3] дается неправильная трактовка двух последних команд.

С внешними устройствами можно общаться посредством специальных ячеек, называемых портами, которые имеют свое адресное пространство и свои команды пересылки. Вот эти команды:

```
IN   AL,15H   ;загрузить в AL байт из порта 15H
IN   AX,DX    ;загрузить в AX слово из порта с номером из DX
OUT  DX,AX    ;аналогично, но из порта в AX
OUT  DX,AL    ;загрузить байт в порт, адресуемый регистром DX
```

и т.п. С портами мы еще встретимся в последующих главах. Описание портов и методов работы с ними см. в Приложении 9.

Со стековыми командами Вы уже познакомились. Не упоминались только команды PUSHF - загрузить в стек регистр флагов, POPF - выгрузить из стека регистр флагов. Интересно, что теперь появляется возможность загрузить свое слово в регистр флагов:

```
PUSH   BX
POPF
```

таким образом, содержимое регистра BX оказывается в регистре флагов. Но будьте с этим осторожны, сначале изучите, какой бит в этом регистре за что отвечает. Существуют еще команды LAHF и SAHF (см. Приложение 1), но я не советовал бы их использовать. Они устарели даже для микропроцессора 8086/8088.

Команда XLAT очень удобна для проведения табличных преобразований. Пусть в DS:BX содержится указатель на некоторую таблицу байт. В AL помещается номер байта в таблице. В результате выполнения команды XLAT в AL будет помещен байт с заданным номером.

II. Команды передачи управления.

К командам передачи управления относятся команды, меняющие регистр указателя команд IP. Другим способом изменить содержимое регистра IP невозможно. Рас-

смотрим сначала команды, не затрагивающие содержимое стека, - это всевозможные переходы. **Команда безусловного перехода JMP имеет три модификации:**

JMP SHORT L1 - короткий переход. Длина команды два **байта.** Поэтому переход осуществляется в пределах 128 байтов (ниже и выше команды).

JMP L1 - длинный переход. Осуществляется в пределах одного сегмента. Длина команды три байта. Возможен также косвенный переход типа JMP [BX]- переход по адресу, который лежит в ячейке, определяемый содержимым регистра BX. Такой переход называется внутрисегментным.

Межсегментный переход. Работу такой команды демонстрирует программа на Рис. 4.1.

```
CODE  SEGMENT
      ASSUME  CS:CODE
      ORG  100H
BEG:
      JMP  SHORT  BEGIN         ;короткий переход
;адрес в ПЗУ, по которому
;находится программа сброса системы
L1     DW OFFFOH          ;смещение
      DW OFOOOH          ;сегмент
BEGIN:
      JMP  DWORD  PTR  L1       ;межсегментный переход
CODE  ENDS
      END  BEG
```

Рис. 4.1. Программа, демонстрирующая межсегментный переход.

После выполнения этой программы возникнет ситуация, аналогичная той, которая создается при нажатии кнопки RESET. Возможен и косвенный межсегментный переход. Замените JMP DWORD PTR L1 на две команды:

```
LEA DI,L1
JMP DWORD PTR CS:[DI]
```

и получившаяся программа будет приводить к тем же результатам, что и предыдущая.

Все команды условного перехода являются короткими. Они приведены в Приложении 1, и мы не будем их перечислять. Разберем лишь одну ситуацию. Рассмотрим следующий фрагмент:

```
      .
      .
      INT 21H
      JC   ERROR ; перейти, если функция DOS выполнилась с ошибкой
      .
      .
      ERROR:
```

В процессе доработки программы количество байт между командой JC и меткой ERROR может увеличиться и превзойти 128 байт. Тогда ассемблер будет давать сообщение об ошибке. Такие ошибки могут доставить Вам много беспокойства, если подобных мест в программе несколько. Обойти эту ситуацию можно следующим образом:

```
        .
        .
        INT  21H
        JNC  NO_ERROR
        JMP  ERROR
NO_ERROR:
        .
        .
ERROR:
```

При выполнении команды условного перехода микропроцессор непосредственно обращается к регистру флагов и в зависимости от состояния соответствующих бит выполняет эту команду. В частности, команды JC и JNC выполняются в зависимости от того, чему равен флаг четности (С, см. главу 3). Циклы на ассемблере можно организовывать как с помощью команд перехода по условию, так и с помощью специализированной команды LOOP. Данная команда использует регистр CX как счетчик числа циклов. На Рис.4.2 показана схема организации вложенных циклов. На Рис.4.2 представлены два вложенных цикла, но по той же системе циклов можно сделать и больше. Не надо только забывать, что LOOP дает короткий переход. Разновидности команды LOOP см. в Приложении 1.

```
        .
        MOV CX,10     ;счетчик внешнего цикла
LOOP1:
        PUSH  CX
        MOV   CX,100  ;счетчик внутреннего цикла
LOOP2:
        PUSH CX
        .
        .
        .
        POP  CX
        LOOP  LOOP2
        POP  CX
        LOOP  LOOP1
        .
        .
```

Рис. 4.2. Организация вложенных циклов.

Рассмотрим теперь команды, связанные с изменением стека. Прежде всего, это команда обращения к подпрограмме (процедуре). Эта команда может длинной - внутрисегментный переход и межсегментный. Длинный CALL кладет в стек адрес возврата (смещение). Возврат из процедуры реализует команда RET. Она извлекает из стека адрес возврата (два байта) и осуществляет длинный переход по указанному адресу. Если процедура имеет атрибут FAR (даже если процедура находится в том же сегменте, что и команда, ее вызывающая), то вызов будет дальним или межсегментным. При этом в стек будет занесен четырехбайтный адрес. Вначале заносится сегментный адрес, а затем смещение. При этом команды RET, стоящие в пределах данной процедуры, становятся межсегментными, т.е. выталкивают из стека четыре байта. У них и отдельная мнемоника RETF. Отсюда Вам должно быть уже ясно, какую роль играют отметки начала и конца процедуры. Итак, RET может оказаться и внутрисегментным и межсегментным, т.е. превратиться в RETF. RETF же всегда RETF. Для внутрисегментного перехода имеется и своя мнемоника RETN. Отмечу также, что если атрибут FAR указать при вызове (CALL FAR PTR SORT), то вызов будет межсегментным, но команды RET, стоящие внутри процедуры, не будут преобразованы к RETF. Следующая программа (Рис. 4.3) демонстрирует работу RETF. Результат ее работы аналогичен результату работы программы на Рис. 4.1.

```
CODE SEGMENT
     ASSUME CS:CODE
BEG:
     JMP SHORT BEGIN
;адрес в ПЗУ, по которому
;находится программа сброса системы
L1    DW OFFFOH    ;адрес в ПЗУ подпрограммы сброса
     DW OFOOOH
BEGIN:
     PUSH L1+2
     PUSH L1
     RETF          ;переход на подпрограмму RESET
CODE ENDS
     END BEG
```

Рис. 4.3. Пример использования RETF вместо JMP.

III. Команды арифметических операций.

Команды арифметических операций сводятся к выполнению четырех арифметических действий: сложения, вычитания, умножения, деления. Команды сложения и вычитания разберем на примерах. На Рис. 4.4 показан фрагмент сложения двух операндов OPER1 и OPER2. Результат помещается в SUM. Проблема заключается в том, что операнды имеют длину 4 байта. Обратите внимание на использование команды ADC. Она автоматически учитывает возникновение переноса из последнего бита (сло-

жение с учетом флага переноса). Само собой разумеется, мы **предполагаем,** что результат **не** будет **превосходить** 4 **байта.**

```
•
•
MOV AX,WORD PTR OPER1       ;поместить первое слово OPER1
ADD AX,WORD PTR OPER2       ;сложить с первым словом OPER2
MOV WORD PTR SUM,AX         ;поместить в первое слово SUM
MOV AX,WORD PTR OPER1+2     ;поместить второе слово OPER1
ADC AX,WORD PTR OPER2+2     ;сложить со вторым словом OPER2
MOV WORD PTR SUM,AX         ;поместить во второе слово SUM
•
•
```

Рис. 4.4. Сложение двух ***четырехбайтных*** *операндов.*

Следующий фрагмент (Рис.4.5) демонстрирует аналогичную работу команд SUB и SBB.

```
•
•
MOV AX,WORD PTR OPER1       /поместить первое слово OPER1
SUB AX,WORD PTR OPER2       ;вычесть первое слово OPER2
MOV WORD PTR SUM,AX         ;поместить в первое слово SUM
MOV AX,WORD PTR OPER1+2     /поместить второе слово OPER1
SBB AX,WORD PTR OPER2+2     /вычесть второе слово OPER2
MOV WORD PTR SUM,AX         ;поместить во второе слово SUM
•
•
```

Рис. 4.5. Вычитание двух четырехбайтных операндов.

Имеется две команды умножения: чисел без знака и чисел со знаком. Читателю должно быть ясно, что алгоритм умножения должен **существенно** измениться, если старший бит считать знаковым. Вот несколько примеров:

```
MUL DX   /умножить DX на AX без знака, результат в DX:AX
IMUL BYTE PTR MEM1  /умножить содержимое ячейки MEM1 на AL со знаком,
                    /результат поместить в AX
MUL BL   ;умножить содержимое регистра BL на AL,  результат в AX.
```

Таким образом в зависимости от того, какого типа операнд стоит в команде умножения (байт или слово), умножение будет произведено над байтом (AL) или над словом (AX). Соответственно результат умножения будет помещен в AX или DX:AX. После исполнения команды MUL флаги **CF** и OF равны **0,** если старшая половина произведения (байт или слово) равна нулю. В противном случае оба флага равны **1.** После **выпол-**

нения команды IMUL флаги CF и OF равны 0, если старшая половина произведения представляет собой лишь расширение знака младшей. В противном случае они равны 1.

Команды деления аналогичны командам умножения. Вот несколько примеров.

```
DIV CX ;разделить DX:AX без знака, результат в AX, остаток в DX
IDIV BYTE PTR MEM1  ;разделить AH:AL на содержимое ячейки MEM1
                    ;со знаком, результат в AL, остаток в AH
DIV DL ;разделить AH:AL на DL без знака, результат в AL, остаток в AH.
```

Команды деления оставляют состояние флагов неопределенным. Если частное не помещается в регистр-приемник, то генерируется прфывание 0 - на экране появляется надпись Divide overflow. При этом дальнейшее выполнение программы прерывается. К арифметическим командам относятся еще INC, DEC, NEG, но с ними, я надеюсь, Вы справитесь сами.

Команды расширения знака позволяют выполнять смешанные действия - байта со словом, слова с двойным словом. CBW воспроизводит 7-й бит регистра AL в регистр AH. CWD воспроизводит 15-й бит регистра AX во всех битах регистра DX.

Особо следует сказать о выполнении арифметических операций над BCD числами. Напомню читателю, что это такое. Упакованное BCD число представляется последовательностью полубайт. Каждый полубайт содержит соответствующую цифру. Например, число 87H в упакованном BCD представлении есть просто 87 (десятичное!). Неупакованный BCD формат представляет собой последовательность байт, где каждый байт представляет одну цифру. Если к каждому такому байту добавить 48, то мы получим фактически цепочку символов, отображающих данное число. Такой формат называют еще ASCII форматом. Микропроцессор, естественно, не знает, над каким числом он производит действие. Для него все числа двоичные. Но есть команды, которые корректируют результат, так что он оказывается правильным. Перечислим эти команды. Коррекция сложения:

```
AAA - для неупакованных чисел,
DAA - для упакованных чисел.
```

Команды выполняются после команды байтового сложения. Предполагается, что результат находится в AL. Коррекция вычитания:

```
AAS - для неупакованных чисел,
DAS - для упакованных чисел.
```

Команды выполняются после команды байтового вычитания. Предполагается, что результат находится в AL. Команда умножения:

```
AAM - для неупакованных чисел.
```

Команда выполняется после команды байтового умножения. Команда деления:

```
AAD - для неупакованных чисел.
```

Команда выполняется до (!) операции деления. Корректирует неупакованное делимое, находящееся в AX, в двоичное число в AL, так чтобы результат оказался правильным.

IV. Операции над отдельными битами.

К операциям над битами мы относим как логические команды типа AND и OR, так и различные сдвиговые команды. Логические команды выполняются побитно. Условно можно считать, что 0 - это ложь, а 1 - истина. Все остальное достаточно тривиально. Например:

```
.
.
MOVAL,0101111B
 MOV DL,1001001B
MOVBL,0001111B
MOVCL,1111000B
  AND AL,DL
  OR   DL,BL
  XOR BL,CL
.
.
```

После выполнения данных команд в регистре AL будет 0001001 B, в регистре DL - 1001111B, в регистре BL - 1110111B. Разберитесь в этом. Команда XOR очень удобна для кодировки. Если в Вашей программе есть текстовые строки, то после трансляции, естественно, эти строки останутся в коде программы. Некоторых может соблазнить исправление этих строк (это можно сделать даже в простом текстовом редакторе, и над моими программами не раз таким образом издевались). В сложившейся ситуации выход достаточно прост - закодируйте часть программы, где находятся строки. При запуске профамма сама раскодирует нужные данные. Кодировку и раскодировку можно произвести с помощью одной и той же команды XOR. Легко проверить, что дважды примененная команда XOR не меняет значения операнда. Команду XOR применяют также для обнуления регистров. Причем команда XOR AX,AX предпочтительнее MOV AX,0, т.к. она короче и быстрее выполняется. Команда OR замечательна тем, что при несовпадении битов двух операндов, она равносильна сложению.

Все сдвиговые операции можно подразделить на три класса - знаковые, беззнаковые (арифметические и логические) и циклические. Сдвиговые операции имеют многочисленное применение, и в дальнейшем я приведу несколько примеров. Здесь же я укажу на наиболее часто используемое применение - использование операций сдвига для умножения и деления. Часто можно написать процедуру, используя команды сдвига, которая будет выполнять умножение или деление быстрее, чем с использованием стандартных команд MUL или DIV. Например, умножение на 16 сведется просто к командам:

```
.
MOV AX,456   /число 456
MOVCL,4      ;сдвиг на четыре бита
SHL AX,CL    ;сдвигаем, т.е. умножаем на 16
.
```

Аналогично можно осуществить деление. В том случае, если делитель или множитель не является степенью числа 2, умножение или деление можно свести к комбинациям сдвиговых операций и операций сложения или вычитания. Предположим, Вы хотите умножить число NUM на 130. Но 130=128+2. Следовательно, фрагмент программы мог бы иметь следующий вид:

```
.
MOVAX,NUM    ;число
MOVCL,7      ;сдвигаем на 7 бит
SHL AX,CL    /умножаем на 128
MOV BX,NUM
SHL BX,1     ;умножаем на 2
ADDAX,BX     ;получили число * на 130
ADDAX,NUM    ;получим окончательный результат
.
```

Еще одна область применения сдвиговых операций - выделение отдельных групп бит в числе. Рассмотрим следующий пример. Предположим, в байте, содержащемся в регистре AL, старшие три бита определяют некоторую величину. Чтобы выделить эту величину удобно использовать сдвиговую операцию. Например, SHRAL,5. Если Вы не хотите потерять и другие биты, то можно действовать следующим образом:

```
.
ROL AL,3            ;3 старших бита стали младшими
MOV AH,AL .         /сохранить число
ANDAL,00000111B     ;выделить биты
SHRAH,3             ;выделяем остальные биты
.
```

V. Строковые команды.

Строковые команды заслуживают того, чтобы остановиться на них более подробно. И хотя все строковые команды приводятся в Приложении, считаю необходимым привести здесь эти команды и дать им характеристики. Прежде всего рассмотрим префиксы повторения. Вот они: REP

```
REPE/REPZ
REPNE/REPNZ
```

Префиксы повторения заставляют микропроцессор повторять строковую команду, пока содержимое регистра CX не станет равным нулю (REP) или пока не произойдет одно из двух событий - CX=0 или флаг Z=1 (Z=0 для префиксов REPNE/REPNZ). Через косую черту перечисляются команды-синонимы. Префиксы REPE/REPZ, REPNE/REPNZ используются только с командами сравнения и поиска (сканирования), префикс REP - с остальными строковыми командами.

Несколько общих замечаний. Строковые команды работают с целыми строками - байт или слов. Строка может быть источником или приемником. Предполагается, что строка-источник находится в сегменте, адресуемом регистром DS со смещением в SI, а строка-приемник находится в сегменте, адресуемом регистром ES, со смещением в DI. При выполнении команды содержимое регистров DI,SI изменяется на 1 (байтовые строки) или на 2 (строки слов). Если флаг DF=нулю, то содержимое индексных регистров увеличивается, если DF=единице, то содержимое индексных регистров уменьшается. Изменить значение флага DF можно с помощью команд CLD - сбросить флаг, STD - установить флаг направления. Каждая из строковых команд имеет по три модификации: с окончанием 'B' байтовая команда, с окончанием 'W' - команда работы со строками слов, без окончания. Команда, не содержащая окончания, при ассемблировании преобразуется к команде с окончанием. О том, к какой команде преобразовывать, ассемблер узнает по наличию в команде операндов. Если операнд определен через DB, то команда становится байтовой, если через DW, то команда оперирует со словами. Например,

```
.
REP MOVS DEST,SOURCE
.
```

Если DEST и SOURCE (одновременно) определены через DW, то команда MOVS сведется к MOVSW. Не рекомендуется использовать команды без окончания, и далее в изложении их не будет. Наконец, замечу, что поскольку счетчик находится в регистре CX, то максимальная длина обрабатываемых строк не должна превышать 64K.

Команды пересылки - MOVSB и MOVSW. В следующем фрагменте происходит пересылка строки байт из SOURCE в DEST. Количество пересылаемых байт равно 200. Во время пересылки значение индексных сегментов увеличивается.

```
.
CLD
MOV CX,200
LEA SI,DS:SOURCE
LEA DI,ES:DEST
REP MOVSB
.
```

Обращаю Ваше внимание на команды загрузки адреса в индексные регистры. В случае с SI можно было явно не указывать сегментный регистр - это подразумевается

по умолчанию. В случае же с DI указание сегмента необходимо. Естественно, что если содержимое сегментных регистров DS и ES совпадает, то пересылка осуществляется в пределах одного сегмента. Причем если строки находятся в дополнительном сегменте (сегменте, на который указывает ES), то изменять содержимое регистра не понадобится. Достаточно написать LEA SI,ES:SOURCE. Однако если обе строки находятся в сегменте данных, то придется менять значение ES.

Команды сравнения строк - CMPSB и CMPSW. Рассмотрим фрагмент:

```
.
STD
MOV CX,100
LEA SI,DS:SOURCE
LEA DI,ES:DEST
REPNE CMPSW
.
```

В представленном примере сравниваются две строки - SOURCE и DEST. Сравнение осуществляется с конца, при этом значение индексных регистров уменьшается. Сравниваются слова - всего 100 слов. Выход из цикла происходит либо при CX=0, либо при нахождении совпадающих элементов. При этом, если были найдены совпадающие элементы, флаг Z=1. Таким образом, используя команду

```
JNZ NO_FOUND ;не обнаружено, если Z=0
```

после указанного фрагмента, мы всегда будем знать, найдены совпадающие элементы или нет. Если такие элементы найдены, то на них будут указывать индексные регистры.

Команды сканирования строк - SCASW и SCASB. Осуществляют поиск элемента в строке. На строку указывает регистр DI. Рассмотрим конкретный пример.

```
.
CLD
MOV CX,200        ;просматриваем 200 слов
LEA DI,STRING     ;DI указывает на строку
MOV AX,7890H      ;слово для поиска
REPNE  SCASW
```

Фрагмент осуществляет поиск в строке слова 7890H. Если слово найдено, то смещение следующего за ним слова будет возвращено в DI. При этом флаг нуля Z будет равен 1.

Команды загрузки и сохранения - LODSB,LODSW,STOSB,STOSW. Команда загрузки LODSB (LODSW) пересылает байт (слово), адресованный SI в AL (AX), а затем изменяет регистр SI согласно флагу направления. Команда сохранения STOSB (STOSB) пересылает байт из AL (AX) в элемент строки, адресуемый DI, затем изменяет содер-

жимое регистра DI согласно флагу направления. Рассмотрим фрагмент копирования строки. Пусть SI указывает на начало строки, DI - на область, куда она копируется, например, экран. В случае экранной памяти следует помнить, что каждому знакоместу соответствует два байта - код символа и его атрибут. Соответственно DS указывает на сегмент, где находится строка, а ES - на сегмент, куда она копируется; CX содержит количество символов в строке:

```
L1:
        LODSB
        STOSB
        LOOP  L1
```

Данные строки полностью эквивалентны следующим при условии соответствующей установки флага направления (DF):

```
L1:
        MOV  AL, [SI]
        MOV ES:[DI],AL
        INC  DI
        INC  SI
        LOOP  L1
```

Завершая разговор о строковых командах, рассмотрим текст следующей программы. Эта программа копирует часть себя за кодовый сегмент (COM-программа) и передает на нее управление. Отработав, фрагмент возвращает управление в основную программу.

```
CODE   SEGMENT
       ASSUME  CS:CODE
       ORG  100H
BEGIN:
       JMP  BEG
TEXT1  DB  'Я за текущим сегментом',13,10,'$'
TEXT2  DB  'Я снова в текущем сегменте',13,10,'$'
WHE    DW ?               ;здесь хранится адрес,
       DW ?               ;с которого будет загружен фрагмент
BEG:
       MOV AX,ES
       ADD AX,1000H       ;максимальная длина сегмента в параграфах
       MOV ES,AX
       MOV DI,0           ;ES:DI на адрес, куда копируем фрагмент
       MOV WHE,DI         ;сохраним
       MOV WHE+2,ES       ;   адрес
       LEA SI,L1          ;смещение копируемого фрагмента
```

```
    MOV CX,L2-L1+1   ;длина копируемого фрагмента
    CLD
REP MOVSB            ; копируем фрагмент
    CALL DWORD PTR WHE ;межсегментный переход на фрагмент
    JMP  L2
;копируемый фрагмент
L1:
    LEA  DX,TEXT1
    MOV  AH,09H
    INT  21H
    RETF                 ;возвращаемся в основную программу
;конец копируемого фрагмента
L2:
    LEA  DX,TEXT2
    MOV AH,09H
    INT 21H
    MOV AH,4CH
    INT 21H
CODE ENDS
    END BEGIN
```

Рис. 4.6. Программа, демонстрирующая работу команды MOVSB.

В начале работы COM-программы все сегментные регистры направлены на текущий сегмент (для EXE-программ это неверно, и мы позднее будем об этом говорить). Внимательно разобравшись в программе, проследите, как в процессе ее выполнения меняется содержимое сегментных регистров и в каких командах используется содержимое того или иного сегментного регистра. Напомню, что в командах загрузки MOV или LEA с любым из регистров, кроме BP, неявно предполагается использование сегментного регистра DS.

VI. Команды управления.

К командам управления относятся, в частности, команды управления флажками. С этими командами легко разобраться по описанию в Приложении. Кратко рассмотрим другие команды.

Команда LOCK - префикс блокировки шины. Используется в многопроцессорной системе и может сопутствовать любой команде микропроцессора. При этом на время работы этой команды закрывается доступ к шине любого другого процессора.

Команда NOP -команда холостого хода. Не производит никаких действий, но счетчик команд изменяется. С помощью этой команды можно, например, забить другие ненужные команды, не прибегая к повторному ассемблированию.

HLT - команда останова. Переводит микропроцессор в состояние останова. Из этого состояния его может вывести лишь внешнее прерывание, например, времени. Последовательность команд CLI/HLT приводит к полному "зависанию" компьютера.

Команда WAIT используется для синхронизации действия микропроцессора с другими устройствами. Смысл этой команды заключается в остановке микропроцессора на то время, пока какое-то другое устройство не закончит выполнять свою работу и перейдет в состояние готовности. Речь в данном случае может идти, например, о математическом сопроцессоре. Прерывания не блокируются, но в отличие от HLT после завершения прерывания микропроцессор возвращается к этой же команде. Микропроцессор и сопроцессор работают параллельно, однако время выполнения разных команд различно. Может случиться, что микропроцессору потребуются результаты вычисления сопроцессора. В этом случае ему придется ждать, когда сопроцессор закончит свою работу. Здесь как раз и нужно воспользоваться командой WAIT.

Команда ESC заставляет микропроцессор передать данные на шину, где этими данными может воспользоваться, например, сопроцессор. Эта команда содержит два операнда. Первым операндом является код команды математического сопроцессора. Вторым - операнд для этой команды. Фактически ESC не является командой микропроцессора, а есть способ записи команды сопроцессора [2,17]. Вместо ESC можно писать команды сопроцессора, используя мнемонические обозначения (см. Приложение 3).

VII. О тестировании микропроцессоров.[10]

Команды, рассмотренные в данном разделе, будут выполняться на всех микропроцессорах серии 8088/8086. Это целое семейство, представителями которого являются микропроцессоры: 8088, 8086, 80188, 80186, 80286, 80386, 80486, Pentium. Добавьте сюда еще микропроцессоры NEC20 и NEC30. Иногда программе требуется знать, с каким процессором ей приходится работать. Конечно, можно попробовать тестировать производительность (мы не будем этим заниматься). Таким образом, Вы легко отличите микропроцессор 8088 от 80286. Однако как различить модели 8086 и 8088?

Более надежным способом, на мой взгляд, является учет нюансов в выполнении команд. О некоторых из них здесь и будет рассказано.

В главе 3 говорилось о буфере команд. Из последовательности команд выбирается несколько (сколько помещается в буфер) и заносится в буфер. По мере выполнения очередь пополняется новыми командами. Так что она всегда полна. Некоторые команды, однако, сбрасывают очередь, так что она заполняется, начиная со следующей команды. К командам, сбрасывающим очередь, относятся INT, IRET, RET, CALL и всевозможные команды перехода. Некоторые марки микропроцессоров можно отличить по размеру буфера. Например, в микропроцессоре 8088 длина буфера 4 байта, а в 8086 - 6. Правда, в микропроцессоре NEC20 длина буфера составляет тоже 4 байта, и здесь потребуется проведение дополнительного тестирования. Ниже (Рис. 4.7) представлена программа, позволяющая определить размер буфера команд. Варьируя количество команд NOP, Вы легко определите длину буфера.

[10] Содержимое данного раздела носит в настоящее время в значительной степени исторический характер.

```
CODE SEGMENT
     ASSUME CS:CODE
     ORG 100H
BEGIN:
     MOV DI,OFFSET MET
     JMP SHORT MET1              ;сбрасываем буфер команд
MET1:
     MOV BYTE PTR [DI],0C3H  ;длина команды 3 байта
     NOP
MET: NOP
;текст будет напечатан, если команда с меткой MET не попадет
;в буфер команд после выполнения команды JMP SHORT MET1
     LEA DX,TEXT
     MOV AH,9
     INT 21H
     RET
TEXT DB 'конец!',13,10,'$'
CODE ENDS
     END BEGIN
```

Рис. 4.7. Определение длины буфера команд.

Еще один признак, позволяющий различать процессоры, это регистр состояния. С появлением новых моделей в нем появляются новые флажки. Не вдаваясь в подробности, рассмотрим основную идею. Вы пытаетесь установить флажок, а потом проверяете его наличие. Для того микропроцессора, где данный флажок отсутствует, соответственно и установить его невозможно. Последовательность команд приблизительно такая:

```
     MOV AX,FLAG
     PUSH AX
     POPF
     PUSHF
     POP BX
     CMP AX,BX
```

Существует некоторое количество "экзотических" команд, выполнение которых может отличаться у разных компьютеров. Примером может служить команда PUSH SP. Вопрос заключается в том, когда из содержимого SP вычитается 2 байта - до того, как содержимое оказалось в стеке, или после. Оказывается, до 286-го микропроцессора значение SP изменялось раньше, чем попадало в стек. Другим критерием, позволяющим судить о том, какой микропроцессор установлен на компьютере, могут служить новые команды. Известно, что команд PUSHA и POPA нет у 8088/8086 микропроцессоров. На Рис. 4.8 представлена программа, использующая данный факт для некоторых выводов о микропроцессоре.

```
.286
CODE   SEGMENT
       ASSUME  CS:CODE
       ORG  100H
BEGIN:
       MOV AX,SP
       PUSHA
       CMP AX,SP
       JNZ  P286
       MOV AH,9
       MOV  DX,OFFSET MES
       INT  21h
       JMP  SHORT KON
P286:
       POPA
KON:
       RET
MES    DB  'Это 86 или 88 процессор',13,10, '$'
CODE  ENDS
       END   BEGIN
```

Рис. 4.8. Программа, использующая команды PUSHA и POPA.

Конечно, ни одна из особенностей микропроцессоров, описанных выше, не может являться исчерпывающим критерием для определения его типа. Они должны использоваться комплексно [15]. Как дополнительный критерий можно использовать скорость выполнения команд микропроцессором, но для этого Вам придется познакомиться с прерыванием по времени. В главе 23 будет дана исчерпывающая программа, определяющая тип микропроцессора.

VIII. Время выполнения команд.

Время выполнения команды микропроцессором измеряется в тактах синхронизации. Зная период синхронизации, можно определить время выполнения команды в обычных единицах. Как правило, говорят о базовом времени - времени выполнения команды, если она находится в буфере команд. Если же команда не находится в буфере команд, то следует учесть количество тактов синхронизации, необходимых для выборки команды из памяти. Если один из операндов команды является ячейкой памяти, то требуется добавить еще время выборки операнда из памяти (время вычисления эффективного адреса). В Приложении 1 приводится количество тактов, требующееся для выполнения каждой команды.

Глава 5. Работа микропроцессора в защищенном режиме.

- Так ты, кум, еще не был у дьяка в новой хате.

Н.В. Гоголь.
Ночь перед рождеством.

I.

До сих пор мы пользовались только командами из набора микропроцессоров 8088/8086. В принципе этот набор будет работать на любом из микропроцессоров данного семейства. С каждым появлением нового микропроцессора расширялся набор команд - к старым командам добавлялись новые. В главе 20 дается полная картина того, как происходил этот процесс. Из всего семейства следует выделить микропроцессоры 80286 и 80386, т.к. они не только расширили набор команд по сравнению с предыдущими, но и явили собой качественный скачок в области программирования. В данной главе мы будем говорить о микропроцессоре 80286, указывая там, где это необходимо, на отличие, появившееся в последующих микропроцессорах.

Микропроцессор 80286 полностью совместим с микропроцессорами 8088/8086. По сути дела, если Вам нет нужды обращаться к защищенному режиму (см. ниже), то можете не думать, на каком микропроцессоре программируете. Считайте, что работе с быстрым 8086-м микропроцессором[1]. В моей книге Вы найдете много программ, которые для своего исполнения требуют микропроцессор, способный выполнять команды 286-го процессора.

Хочу подчеркнуть, что мы не будем рассматривать какие-либо технические характеристики микропроцессоров, изложение будет ограничено только программными рамками. Но и это в силу большого объема материала будет изложено конспективно кратко. Отсылаю всех интересующихся к книгам [18, 22].

Микропроцессор 80286 может работать в двух режимах: реальном и защищенном. Реальный режим представляет фактически эмуляцию работы микропроцессоров 8088/8086 с увеличением быстродействия и некоторого расширения множества команд. Защищенный режим качественно отличается от реального режима поддержкой многозадачных операционных систем.

Рассмотрим вначале некоторые команды реального режима, которые могут быть полезны в программировании.

---

[1] Была еще промежуточная модель микропроцессоров этой серии 80186. Но выпуск этой модели не был слишком массовым, и вряд ли Вы встретите компьютер на таком микропроцессоре. Лично я не встречал (см. главу 20). Говорить же о быстроте 286-го микропроцессора во времена Pentium'ов, конечно, можно лишь относительно.

II.

Сделаем краткий обзор команд микропроцессора 80286 реального режима (новых по отношению к микропроцессору 8086). Интересно, что, несмотря на то что сейчас повсеместно уже используются Pentium'ы, многие программисты, пишущие на ассемблере, почему-то забывают об этих командах.

Расширение команд коснулось стека. Прежде всего появились команды PUSHA - все регистры в стек и POPA - все регистры из стека[12]. Очень эффективные команды, особенно для создания процедур. Кроме того, появилась возможность отправлять в стек непосредственный операнд, например, PUSH 457H (в стек помещается слово 0457H).

Расширена команда знакового умножения. Теперь появилась возможность выполнять команды: IMUL DX,BX,123 (BX*123 --> DX) или IMUL CX,MEMW,567 (MEMW*567-->CX). Здесь MEMW - ячейка памяти. Арсенал логических команд дополнен командой NOT, которая инвертирует все биты операнда. Расширены сдвиговые команды. Стали возможны команды типа SHL AX,7 или SHR MEM, 10. В микропроцессоре 8086/8088 для выполнения подобных действий потребовалось бы две команды: MOV CL,7/SHL AX,CL.

К цепочечным (строковым) командам добавлено две команды: OUTS - запись данных из последовательных ячеек памяти в выходное устройство; INS - считывание данных из входного устройства в последовательные ячейки памяти. Они упрощают передачу больших блоков данных между памятью и внешними устройствами. Примитив ввода INS передает данные из входного порта, определяемого содержимым DX, в байт или слово, смещение которого находится в DI, и производит инкремент (декремент) содержимого DI. Аналогично OUTS передает байт или слово из памяти в порт.

Среди команд условного перехода появилась команда JCXZ - передать управление, если CX содержит 0.

В перечне команд микропроцессора 80286 имеются три команды, упрощающие работу компиляторов языков высокого уровня. Рассмотрим их.

Команда BOUND проверяет, находится ли знаковое значение из 16-битного регистра в заданных пределах. Пределы находятся в двух смежных словах памяти. Значение в регистре должно быть больше или равно значению в первом слове и меньше или равно значению во втором слове; в противном случае генерируется некоторое прерывание. Например, команда BOUND AX,MEMW проверяет, находится ли слово из AX в промежутке, определяемом смежными словами MEMW и MEMW+2. Эта команда может быть удобна для проверки того, находится ли индекс массива в заданных пределах.

Оставшиеся две команды ENTER и LEAVE называют процедурными. Команда ENTER OP1,OP2 готовит стек для входа в процедуру. OP1 показывает количество байт для локальных переменных в процедуре, OP2 - уровень вложенности в процедуру. При OP2=0 вложенность процедур не допускается (ситуация в языке Си). Команда LEAVE не имеет аргументов, она освобождает стек, занятый командой ENTER.

---

[12] Справедливости ради следует сказать, что все команды, описанные до раздела III, были уже в микропроцессоре 80186.

Например, команда:

ENTER N,0 (обычная при входе в функцию Си) эквивалентна последовательности команд PUSH BP\MOV BP,SP\SUB SP,4, а команда LEAVE - последовательности MOV SP,BP\POP BP. (см. Главу 15).

Если Вам необходимо использовать указанные команды в своей программе, не забудьте в начале программы поставить опцию .286 либо .286P, если собираетесь использовать команды защищенного режима (см. ниже). У компиляторов языков высокого уровня есть соответственно опции для трансляции в команды микропроцессора 80286. Причем в этом случае код программы будет более компактным и быстрым.

III. Вычисление физического адреса в защищенном режиме.

Адрес в микропроцессорах 8088/8086 и 80286 в реальном режиме состоит из сегментного адреса (селектора) и смещения (см. главу 2). Реальный физический адрес получается умножением сегментного адреса на 16 и добавлением к нему смещения. В результате получается 20-битный физический адрес (20-разрядная шина адреса). В защищенном же режиме виртуальный (логический) адрес также состоит из селектора и смещения, но физический адрес получается теперь не сдвигом сегментного адреса, а путем индексирования - обращению к таблице адресов. Шина адресов для 286-го процессора состоит из 24 разрядов (из 32 для 386-го). Селектор (содержимое сегментного регистра) состоит из трех полей:

13 бит	*1 бит*	*2 бита*
И н х s к с	*m*	RPL

RPL - запрашиваемый уровень привилегий, используется операционной системой для поддержки разделенного доступа;

Т - индикатор,

Т=0 - индекс указывает на глобальную дескрипторную таблицу (GDT) - GLOBAL DESCRIPT TABLE.

Т= 1 - индекс указывает на локальную дескрипторную таблицу (LDT) - LOCAL DESCRIPTTABLE.

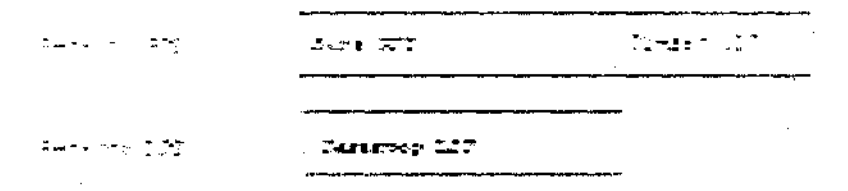
Каждая задача (в узком смысле слова - программа) имеет свою LDT. Базовые адреса сегментов, разделяемых всеми задачами, хранятся в GDT. Базовые адреса "частных" сегментов каждой задачи хранятся в LDT. Элемент таблицы называется дескриптором, нумерация этих элементов начинается (индексируется) с нуля. Дескриптор имеет длину восемь байт, три байта из восьми определяют адрес сегмента (24 бита с объемом адресации 16М). Полученный из таблицы адрес суммируется со смещением, и полученный 24-битный результат является результирующим адресом. Случай с Т=0 и равным нулю индексом называется особым - дескриптор с нулевым индексом - пустым дескриптором. Пустые селекторы не обращаются к дескриптору в GDT, а вызывают особый случай, обрабатываемый особым образом.

Так как длина поля индекса в селекторе составляет 13 бит, то в дескрипторной таблице может быть самое большее 2^13 дескрипторов. Каждый дескриптор описыва-

ет сегмент, длиной до 2^16 байт. Каждая задача может иметь, таким образом, в своем распоряжении 2^13*2^16байт. Так как аппаратно допускается память в 2^16 байт, для использования всего допустимого адресного пространства используются внешние запоминающие устройства. Реально операционная система может сбрасывать часть сегментов на диск и по мере необходимости считывать их в память. Кроме того, в системе имеется таблица дескрипторов прерываний IDT (см. ниже и главу 26).

IV. Регистры преобразования адреса.

Ниже на рисунке представлены регистры микропроцессора GDT и LDT, участвующие в преобразовании адреса.



Регистр GDT содержит физический адрес таблицы GDT (24 бита) и ее длину (16 бит). Выход за длину вызывает особый случай (защита). Регистр LDT (16 бит) хранит селектор сегмента, содержащего текущую LDT. Этот сегмент индексирует дескриптор в GDT. Во время переключения задач микропроцессор изменяет все адресное пространство перезагрузкой регистра LDT.

Загрузка регистров GDT и LDT из памяти (или регистра) и сохранение содержимого этих регистров в памяти осуществляется командами LGDT, LLDT и SGDT, SLDT соответственно (см. ниже).

Регистр GDT обычно загружается во время инициализации системы и в дальнейшем его содержимое не меняется. Локальные дескрипторные таблицы используются в многозадачных операционных системах. Если в защищенном режиме работает лишь одна задача, то ей нет необходимости создавать локальные дескрипторные таблицы, а поэтому нет необходимости как-то использовать регистр LDT.

Параллельно регистру LDT и сегментным регистрам (CS, DS, SS, ES) существуют теневые сегментные регистры. В них запоминаются промежуточные результаты вычисления физического адреса, которые потом используются, ускоряя обращение к памяти. Ниже приводится алгоритм вычисления физического адреса по селектору и смещению:

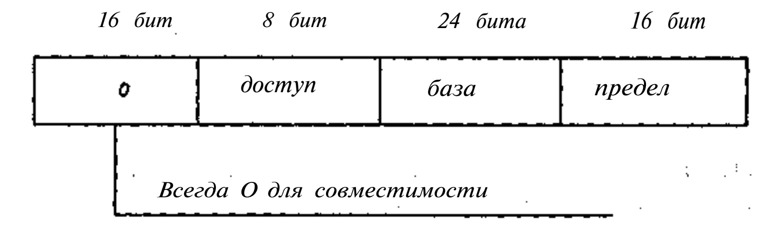
1. Взять селектор из сегментного регистра.
2. Если бит TI показывает на GDT, то взять из регистра GDT адрес дескрипторной таблицы и перейти к шагу 4.
3. Если бит TI показывает на LDT, то
   а) взять селектор сегмента LDT из регистра LDT;
   б) выделить в селекторе поле индекса и умножить индекс на 8. Это объясняется тем, что длина дескриптора составляет 8 байт;
   в) прибавить результат к адресу GDT из регистра GDT;
   г) считать из памяти адресуемый дескриптор;

д) выделить из этого дескриптора базовый адрес сегмента, содержащего таблицу LDT. Полученный базовый адрес является адресом дескрипторной таблицы. Перейти к шагу 4.

4. Выделить значение из поля индекса селектора, умножить его на 8 и прибавить к адресу дескрипторной таблицы. Считать из памяти дескриптор по этому адресу.
5. Выделить из дескриптора базовый адрес сегмента.
6. Прибавить значение смещения к базовому адресу сегмента. Это и будет физический адрес.
7. Осуществить обращение к памяти по физическому адресу.

V. Дескриптор сегмента.

Ниже представлен формат дескриптора сегмента.



Байт доступа имеет структуру, представленную на Рис. 5.1.

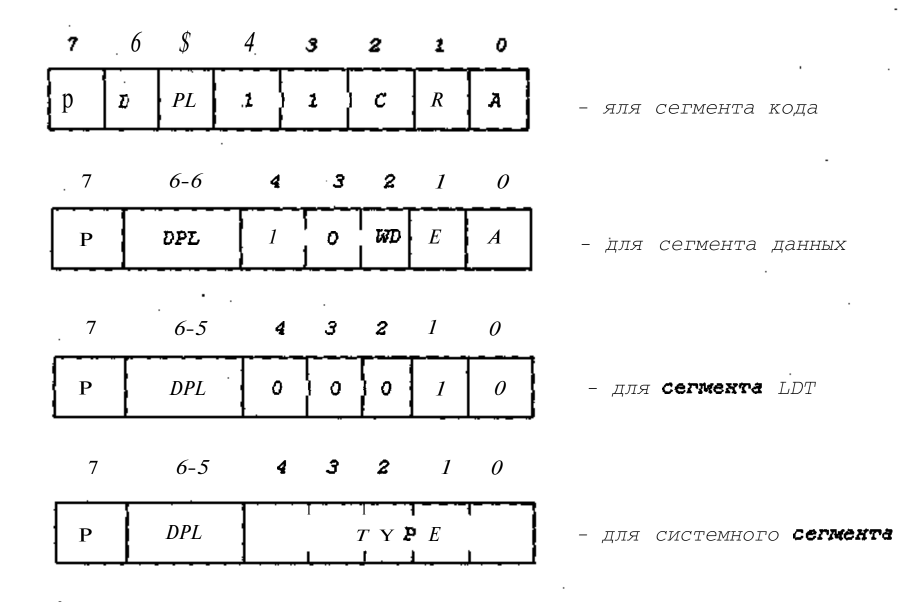


Рис. 5.1. Форматы байта доступа дескриптора.

Байт доступа закодирован так, что путем его анализа всегда можно однозначно определить тип дескриптора. Ниже приводится расшифровка полей байта доступа.

DPL - уровень привилегий, предотвращает доступ обычных задач к сегментам операционной системы.

P - бит наличия. Определяет, находится сегмент в памяти или нет (0 -в памяти, 1 - на диске). В том случае, если места в памяти нет, то какой-то сегмент следует сбросить на диск.

A - бит обращения. Помогает выбрать оптимальную стратегию сброса и подкачки (с диска) сегментов. A обращается в 1, когда селектор для дескриптора загружается в сегментный регистр. Сброс A в 0 осуществляет программа. Бит A т.о. приблизительно показывает, было ли обращение к сегменту после того, как программа сбросила его в 0. Первоначально биты A равны 0. Через регулярные интервалы времени операционная система просматривает дескрипторы и сбрасывает его в 0. Если обнаруживаются дескрипторы с A= 1, то к ним после сканирования были обращения. Т.о. можно выявить сегменты с более частым к ним обращением.

R - бит считываемого сегмента. Если R= 1, то наряду с обращением к такому сегменту для выполнения разрешено обращение для считывания. Если R=0, то попытка считывания вызывает особый случай защиты. Попытка записать в исполняемый сегмент также вызывает особый случай защиты.

W - бит разрешения записи. Если W= 1, то, помимо считывания, возможна и запись. При W=0, попытка записать в сегмент вызывает особый случай защиты.

С сегментом LDT нельзя выполнять операции считывания, записи или выполнения.

C - бит подчинения.

ED- бит расширения. Он применяется для отметки сегментов стека, которые расширяются в область меньших адресов. Он управляет интерпретацией поля предела дескриптора. При ED=1 сегмент предназначен для стеков.

Для системных сегментов первые 4 бита определяют его тип:

0 - запрещенное значение

1 - доступный TSS [13] для процессора 80286

2 - сегмент LDT (см. Рис. 5.1)

3 - занятый TSS для процессора 80286

4 - вентиль вызова для процессора 80286 (см. ниже)

5 - вентиль задачи для процессоров 80286 и 80386

6 - вентиль прерывания для процессора 80286

7 - вентиль исключения для процессора 80286

8 - запрещенное значение

9 - доступ TSS для процессора 80386

A - зарезервировано

B - занятый TSS для процессора 80386

C - вентиль вызова для процессора 80386

D - зарезервировано

E - вентиль прерывания для процессора 80386

F - вентиль ловушки для процессора 80386

---

[13] TSS - Task State Segment - сегмент состояния задачи, т.е. сегмент, в котором сохраняется содержимое регистров задачи, когда процессор выполняет другую задачу.

Продолжим определение других частей дескриптора.

Предел - содержит размер сегмента в байтах, уменьшенный на 1.

База - физический адрес сегмента длиной 24 бита.

Последние два байта (16 бит) содержат для 286-го процессора всегда 0. Для 386-го процессора там, в частности, хранится старший байт 32-битного адреса.

Как видим, адресное пространство для 286-го процессора расширяется до размеров 2 в 24 степени, т.е. до 16 Мб. Для 386-го процессора адресное пространство увеличивается еще в 256 раз (2 в 8 степени), т.е. составляет 4 Гб.

Таблица прерываний в защищенном режиме состоит из дескрипторов прерываний. Ниже приводится структура дескриптора такой таблицы. Дескрипторы таблицы прерываний называются еще вентилями.

16	9	8	16	16
Резерв	Доступ	Парам	Селектор	Смещение

Селектор и смещение показывают на процедуру обработки прерывания. Старшие 16 бит используются 32-битными микропроцессорами для получения 32-битного адреса. Поле Парам используется для передачи параметров вызываемому модулю.



7	6-5	4	3	2	1	0
P	DPL	0	0	1	1	0

Поле доступа вентиля прерывания

7	6 - 5	4	3	2	1	0
P	DPL	0	0	1	1	1

Поле доступа вентиля исключения

7	6-5	4	3	2	1	0
P	DPL	0	0	1	0	1

Поле доступа вентиля (шлюза) задачи

Расположение таблицы прерываний определяется регистром прерываний (IDT). Структура этого регистра аналогична структуре регистра GDTR. Загрузка регистра осуществляется командой LIDT.

Добавление для 386-го микропроцессора.

Кратко отметим изменения, произведенные в 386-го микропроцессоре. Более подробно см. главы 20, 26.

Рассмотрим содержимое старших 16 бит дескриптора для 386-го микропроцессора.

Дескриптор сегмента.

Биты с 48 по 51 расширяют значение предела, т.е. длины сегмента. Т.о. длина сегмента определяется теперь 20 битами и может, следовательно, составлять 1 Мб.

Бит 52 - бит пользователя, предназначен для использования системным программистом.

Бит 53 - зарезервирован для будущих микропроцессоров.

Бит 54 - если содержит 0, то находящиеся в сегменте операнды интерпретируются как 16-битные, в противном случае они считаются 32-битные.

Бит 55 - бит гранулярности. Определяет в байтах или в страницах будет измеряться длина сегмента.

Биты с 56 по 63 расширяют значение базового адреса. Длина базового адреса, таким образом, становится 32-битной.

Дескриптор таблицы LDT.

Отличается от предыдущего только тем, что биты с 52 по 55 равны нулю.

Дескриптор прерывания.

Последние 16 бит расширяют значение смещения до 32 бит.

VI. Структура разделения между задачами.

В однопользовательской системе, где не требуется изолирование задач, можно поместить пустой селектор в регистр LDT для каждой задачи. Тогда каждая задача будет разделять одно адресное пространство, и операционная система может не строить никаких LDT. Такой режим удобен для небольших систем.

Группа взаимосвязанных задач может использовать одну и ту же LDT. Все задачи в группе разделяют одно и то же адресное пространство, но группа как целое обладает частными сегментами, изолированными от других. Такую группу задач называют заданием.

Более сложные структуры разделения и изолирования можно построить, обеспечив каждую задачу ее собственной LDT, но допустив несколько дескрипторов для каждого сегмента. Чтобы разделить сегмент между двумя задачами, каждая задача должна иметь идентичный дескриптор сегмента в своей LDT.

Для каждой задачи существует свой сегмент состояния задачи (TSS). В нем сохраняется содержимое регистров текущей задачи, когда происходит переключение на новую задачу. Информация о сегменте состояния (селектор) задачи содержится в регистре состояния задачи (TR). Регистр состояния задачи загружается после перехода в

защищенный режим. В дальнейшем его содержимое меняется автоматически при переходе от одной задачи к другой.

Операционная система в специальном сегменте хранит селекторы всех выполняемых задач. Периодически обращаясь то к одному, то к другому селектору (переключая задачи), операционная система осуществляет многозадачный режим работы системы.

Рассмотрим немного подробнее вопрос о переключении задач. Как мы уже говорили, состояние задачи хранится в специальном сегменте TSS. Селектор этого сегмента текущей задачи содержится в регистре TR. Ниже представлена структура сегмента состояния задачи.

Обратная связь
Начало стека, кольцо 0
Начало стека, кольцо 1
Начало стека, кольцо 2
IP
Флажки
AX
CX
DX
BX
SP
BP
SI
DI
ES
CS
SS
DS
Регистр LDT
...

Обратите внимание на поле "Обратная связь". В многозадачных системах это поле используют как указатель на следующий TSS. Таким образом, образуется структура в виде списка. Работая с этой структурой ОС система легко может переключаться с одной задачи на другую. Переключение осуществляется путем межсегментного косвенного перехода (команда JMP).

VII. Команды защищенного режима.

Перечислим команды защищенного режима с краткой их аннотацией.

LGDT - загружает GDT из памяти. При загрузке передается шесть байт. Первые пять байт загружаются в регистр GDT, шестой байт игнорируется (загружается в 386-м процессоре). Используется при инициализации (см. главу 26).

SGDT - передает содержимое регистра GDT в адресуемую область памяти. Команда используется в системных отладчиках.

LLDT - загружает операнд-слово из регистра или памяти в регистр LDT. Данное слово будет являться селектором, определяющим выбор локального дескриптора из таблицы GDT. Используется при инициализации.

SLDT - передает содержимое регистра LDT в операнд-слово. Команда, обратная LLDT.

LAR - команда загружает в свой первый операнд байт доступа дескриптора, выбираемый вторым операндом.

LSL - действует аналогично LAR, но загружает в свой первый операнд значение предела выбранного дескриптора.

LMSW - загружает из памяти или регистра регистр состояния. Используется для перехода в защищенный режим. Например, LMSW AX.

SMSW - команда обратная предыдущей.

LTR - загрузка регистра задачи. Выполняется один раз после перехода в защищенный режим.

CLTS - сброс флага переключения задачи.

LIDT - загрузка регистра таблицы прерываний.

VIII. Инициализация системы.

Запуская микропроцессор 80286 в реальном режиме, мы можем затем задать все таблицы и регистры, необходимые для защищенного режима, а потом перейти в защищенный режим. Процессор переключается из реального режима в защищенный командой LMSW, которая загружает в регистр состояния микропроцессора (не путать с регистром флагов) слово, в котором бит разрешения PE равен 1 (бит 0). Ясно, что переход возможен только после инициализации нужных регистров (GDT, LDT и т.д.) и таблиц (GDT, LDT и др.).

Ниже приводится фрагмент установки бита PE в 1. Заметим, что при выполнении данного фрагмента другие биты регистра состояния не изменяются.

```
SMSW   AX
OR     AX, 1
LMSW   AX
```

В регистре состояния микропроцессора используются еще 3 бита, вот они:

бит 1 - должен быть восстановлен при инициализации, если имеется сопроцессор (80287),

бит 2 - восстанавливается в случае отсутствия сопроцессора,

бит 3 - используется при переключении задач (признак переключения).

Остальные биты 286-м микропроцессором не используются.

Рассмотрим теперь последовательность шагов, необходимых для инициализации защищенного режима:

1. Подготовить в оперативной памяти глобальную таблицу дескрипторов GDT. Впоследствии в защищенном режиме можно модифицировать GDT, лишь находясь в нулевом кольце защиты (см. ниже).
2. Запретить все маскируемые и немаскируемые прерывания.
3. Если предполагается использовать в защищенном режиме расширенную память, то следует открыть адресную линию А20 (см. [19, 22]).
4. Загрузить регистр GDTR.
5. Подготовить дескрипторную таблицу прерываний и загрузить регистр IDTR. Эта таблица постоянно активна в защищенном режиме и направляет центральный процессор к программам обработки прерываний в случае соответствующей команды вызова или возникновения особого случая.
6. Если в защищенном режиме предполагается переключение задач, то следует перед инициализацией создать локальную дескрипторную таблицу и загрузить регистр LDTR. Конечно, это можно сделать и после инициализации, находясь в нулевом кольце защиты.
7. Установить бит РЕ и тем самым перейти в защищенный режим.
8. Сбросить очередь команд.
9. Загрузить регистр задачи (команда LTR).

Далее в главе 26 мы покажем реализацию данного алгоритма.

Функция 89Н прерывания 15Н также позволяет перейти в защищенный режим (см. ниже).

IX. Кольца защиты.

Для защиты сегментов в микропроцессоре 80286 принята схема уровней привилегий. Система привилегий регулирует доступ к тому или иному сегменту в зависимости от уровня его защищенности и от привилегированности запроса. Защита требует наложения на обычные программы (невходящие в операционную систему) трех типов ограничений:

1. обычным программам запрещается выполнять некоторые команды;
2. обычным программам должны быть недоступны определенные сегменты, доступные операционной системе;
3. должно быть невозможным получение привилегии операционной системы, кроме как входом в нее в разрешенной точке входа.

Уровни привилегий определяются полями DPL дескриптора. Поскольку на DPL отводится 2 бита, то имеется 4 уровня привилегий. Например, если код находится в сегменте с DPL 00, то это самый высокий уровень - имеет доступ ко всем сегментам, но к этому сегменту имеет доступ только код с сегментом, у которого дескриптор имеет уровень привилегий 00.

Наименее защищенными являются прикладные программы пользователя, для которых выделяется уровень с номером 3. Уровни с номерами 0, 1, 2 отводятся для системных программ. Наиболее защищенная часть - ядро операционной системы имеет

уровень привилегий 0. Основная же часть программ операционной системы имеет уровень привилегий 1. Наконец уровень 2 обычно имеют ряд служебных программ.

Устанавливаются следующие правила доступа:

1. Данные из сегмента могут быть выбраны программой, имеющей такой же или более высокий уровень привилегий.
2. Сегмент программы или процедуры может быть вызван программой, имеющей такой же или более низкий уровень привилегий.

Из правила 2 есть исключение. Командой CALL можно вызывать некоторые процедуры с более высоким уровнем привилегий, используя специальные точки входа (шлюзы). Эти шлюзы имеют свои дескрипторы, которые хранятся в LDT. Шлюзы, в частности, используют для предоставления возможности прикладным программам использовать ресурсы операционной системы. Задача может вызвать данную процедуру или задачу через шлюз, если ее привилегия равна или выше привилегии шлюза (но не задачи, на которую данный шлюз указывает). Таким образом, операционная система может регулировать доступ к некоторым ее внутренним задачам и процедурам.

Для полноты защиты часть команд реального режима должна выполняться только в нулевом кольце. Это команды ввода-вывода: OUT, IN, OUTS, INS. В частности, с помощью этих команд программа посредством контролера прямого доступа к памяти (ПДП) сможет получить доступ к любому сегменту. Кроме этого, в кольцах с ненулевой привилегией запрещены команды, способные заблокировать прерывания: CLI, STI, LOCK.

X. Прерывание 15H.

Данное прерывание доступно лишь на АТ-компьютерах и, в частности, позволяет работать с защищенным режимом. Таковой является функция 89H данного прерывания. Эта функция переводит микропроцессор в защищенный режим. Данной функцией пользуются многие программы, работающие в защищенном режиме.

Пример использования ее можно найти в книге [19]. Заметим также, что BIOS не дает возможности переходить из защищенного режима в реальный, в первую очередь по причине невозможности использовать старые векторы прерывания в защищенном режиме в их старом качестве.

Мы же остановимся здесь на двух других функциях данного прерывания, позволяющих работать с расширенной памятью. Подчеркну лишний раз, что данная проблема актуальна лишь для операционной системы MS DOS.

Как известно, стандартный объем ОЗУ, который имеется у современного IBM-совместимого компьютера и которым можно пользоваться обычным способом (например, MOV ES:[BX],AL) составляет 640 Кб. (см. главу 2). Остальное доступное адресное пространство отведено под адреса ПЗУ и видеопамять (см. главу 22). Отсюда следует, что добавленная оперативная память получит адреса за одним мегабайтом (ее называют расширенной). Ясно, что обычными способами нам до этой памяти не добраться. В защищенном режиме, однако, можно адресовать до 16 Мб (пока речь идет лишь о 286 процессоре). То есть доступ к расширенной памяти можно осуществить через защищенный режим. Именно таким способом функция 87H прерывания 15H осуществляет доступ к памяти за 1 Мб.

Для того чтобы узнать объем расширенной памяти используется функция 88Н:

АН-88Н

Выход:

АХ - объем в килобайтах непрерывного участка расширенной памяти, начиная с адреса 100000Н(1048576). Если взведен флаг переноса, то данная функция не поддерживается.

Заметим, что многие драйверы ОС MS DOS, работающие с расширенной памятью, блокируют работу прерывания 15Н. При этом функция 88Н будет давать 0 килобайт. К таким драйверам относится, в частности, известный HIMEM.SYS.

Теперь самым подробным образом опишем функцию 87Н, позволяющую получить доступ к расширенной памяти.

Вход:

АН - 87Н,

СХ - размер перемещаемого блока в словах, максимум составляет число 8000Н, что соответствует 64 Кб.

ES:SI - указывает на список дескрипторов (таблицу GDT), данная таблица необходима для перехода и работы в защищенном режиме.

Выход:

если флаг переноса не взведен, то в АН - 0, если взведен, то

АН - 1 ошибка схем контроля,

2 выполнение прекращено,

3 неверный адрес памяти.

Рассмотрим подробно структуру таблицы GDT:

- пустой дескриптор, устанавливается на адрес 0, дескриптор данной таблицы дескрипторов, устанавливается в 0, модифицируется BIOS,
- дескриптор перемещаемого блока памяти,
- дескриптор области, куда будет перемещен блок,
- дескриптор кодового сегмента программы, устанавливается в 0, модифицируется BIOS,
- дескриптор стека программы, устанавливается в 0, модифицируется BIOS.

Таким образом, пользователь устанавливает в значение, отличное от 0, только третий и четвертый дескрипторы.

Вспомним теперь структуру дескриптора (см. раздел IV данной главы): вначале (16 бит) идет размер описываемого сегмента. Легко сообразить, что это должно быть значение не меньшее 2*СХ-1, далее (24 бита) - физический адрес блока, следом идет байт доступа, с учетом изложенной выше схемы байт доступа следует выбрать равным 93Н, наконец, последние два байта должны быть равны нулю.

```
.286
WHERE  EQU  ЮН        ;старший байт начала расширенной памяти
DATA  SEGMENT
;--начало GDT1 для копирования в расширенную память
INT_15_GDT  LABEL  BYTE
```

```
;16 нулевых байт
     DB 8 DUP(O)
     DB 8 DUP(O)
;предел сегмента = CX*2-1
     DW 31
;24-разрядный адрес исходного блока
SRC_LO DW ?
SRC_HI DB ?
; байт доступа
     DB 93H
;нулевое поле
     DW O
;предел сегмента = CX*2-1
     DW 31
;24-х разрядный адрес результирующего блока
DST_LO DW ?
DST_HI DB ?
;байт доступа
     DB 93H
;нулевое поле
     DW O
;16 нулевых байт
     DB 8 DUP(O)
     DB 8 DUP(O)
;--конец GDT1
;--начало GDT2 для копирования из расширенной памяти
INT_15_GDT1 LABEL BYTE
;16 нулевых байт
     DB 8 DUP(0)
     DB 8 DUP(0)
;предел сегмента = CX*2-1
     DW 31
;24-разрядный адрес исходного блока
SRC_L01 DW ?
SRC_HI1 DB ?
;байт доступа
     DB 93H
/нулевое поле
     DW O
/предел сегмента = CX*2-1
     DW 31
;24-разрядный адрес результирующего блока
DST_L01 DW ?
DST_НИ DB ?
```

```
;байт доступа
     DB 93H
;нулевое поле
     DW 0
;16 нулевых байт
     DB 8 DUP(0)
     DB 8 DUP(0)
;--конец GDT2
DATA ENDS
;сегмент для пересылаемых или принимаемых данных
;вначале здесь содержится строка, состоящая из букв 'A'
DATA1  SEGMENT
     DB 32 DUP(65)
     DB 13,10,'$'
DATA1  ENDS
;сегмент стека
ST1 SEGMENT STACK
     DB 200 DUP(?)
ST1 ENDS
CODE SEGMENT
     ASSUME CS:CODE, DS:DATA, SS:ST1
BEGIN:
;часть I
;--установка регистров
     MOV AX,DATA
     MOV DS,AX
     MOV ES,AX
     LEA SI,DS:INT_15_GDT
;--установка GDT
;вначале откуда
     MOV  AX,DATA1
     MOV  DL,AH
     SHL  AX,4          ;*
     SHR  DL,4          ;*
     MOV  SRC_LO,AX
     MOV  SRC_HI,DL
;теперь куда
     MOV  DST_HI,WHERE
     MOV  DST_LO,00000H
;теперь запуск
     MOV  CX,16
     MOV  AH,87H
     INT  15H
;проверка выполнения
```

```
    MOV  DL,AH
    MOV  AH, 2
    INT  21H
;теперь выводим строку
    MOV  AX,DATA1
    MOV  DS,AX
    MOV  DX, 0
    MOV  AH, 9
    INT  21H
;заменяем на строку, состоящую из букв 'B'
    MOV  CX,32
    MOV  BX,DATA1
LL:
    MOV  BYTE PTR DS:[BX],66
    INC  BX
    LOOP LL
;часть II
;--установка регистров
    MOV  AX,DATA
    MOV  DS,AX
    MOV  ES,AX
    LEA  SI,DS:INT_15_GDT1
;--установка GDT
;вначале куда
    MOV  AX,DATA1
    MOV  DL,AH
    SHL  AX,4          ; *
    SHR  DL,4           ;*
    MOV  DST_LO1,AX
    MOV  DST_HI1,DL
;теперь откуда
    MOV  SRC_HI1,WHERE
    MOV  SRC_LO1,00000H
;теперь запуск
    MOV  CX,16
    MOV  AH,87H
    INT  15H
;проверка выполнения
    MOV  DL,AH
    MOV  AH, 2
    INT  21H
;теперь выводим строку
    MOV  AX,DATA1
    MOV  DS,AX
```

```
        MOV   DX,0
        MOV   AH,9
        INT   21H
EXIT:
        MOV AH,4CH
        INT 21H
CODE  ENDS
        END  BEGIN
```

Рис. 5.2. Программа демонстрирует копирование данных из обычной памяти в расширенную и обратно.

На Рис. 5.2 представлена программа, демонстрирующая копирование в расширенную память и обратно. Как видим, механизм работы с расширенной памятью весьма прост, и Вы легко сможете написать свои процедуры работы с расширенной памятью. При этом следует учесть, что при выполнении обмена данными с расширенной памятью отключаются все прерывания, поэтому не рекомендуется копировать слишком большие объемы информации. Отметим, что в программе команды 286-го процессора отмечены звездочками.

В заключение скажу, что более подробную информацию о работе с расширенной, дополнительной памятью, а также памятью HMA и UMB можно найти в главе 22.

Глава 6. Уровни программирования.

- Я люблю сидеть низко, - заговорил артист, - с низкого не так опасно падать.

М.А. *Булгаков*
Мастер и Маргарита.

I.

Будем различать три уровня программирования устройств в операционной системе MS DOS и дадим следующие их определения. Высокий уровень - программирование с помощью системных вызовов (функций прерывания 21H, а также прерываний 25H, 26H и других). Средний уровень - программирование при помощи функций BIOS (функции прерываний 10H, 13H, 15H, 16H и др.). Низкий уровень - программирование посредством обращения к аппаратуре через соответствующие порты ввода-вывода или другим способом. Надо сказать, что в литературе можно встретить и другое определение уровней программирования. Так, например, в [5] также выделяются три уровня программирования: программирование на языке высокого уровня, программирование с помощью функций DOS и BIOS и непосредственное программирование устройств.

Выбор уровня программирования влияет на переносимость программ с одного типа компьютера на другой. Высокий уровень обеспечивает наиболее полную совместимость. Здесь, однако, следует помнить о различных версиях операционной системы. Следуйте простому правилу - не пользуйтесь недокументированными возможностями: они могут измениться в последующих версиях. Это происходит не слишком часто, чаще происходит наоборот: недокументированные возможности становятся документированными. С другой стороны, некоторых эффектов можно достигнуть только при низком уровне программирования. Для нестандартных же устройств только низкий уровень программирования дает желаемый результат [14].

Прежде чем приступить к конкретным примерам, познакомимся более подробно с операционной системой MS DOS. Загрузка операционной системы происходит с дискеты или с активного раздела жесткого диска. Чтобы загрузка прошла успешно нужно:

1. В первом секторе дискеты (раздела) должна содержаться загрузочная запись - маленькая программа, осуществляющая начальную загрузку операционной системы.
2. Файлы MSDOS.SYS и IO.SYS, расположенные в фиксированных областях дискеты (раздела).

---

[14] Следует отметить, что MS DOS дает широкие возможности только для работы с файлами. Все остальное представлено довольно скудно. Это, несомненно, вынуждает программистов самим программировать различные внешние устройства, что увеличивает время программирования и ухудшает совместимость программных продуктов. Операционная система Windows лишена данного недостатка, там все внешние устройства программируются средствами операционной системы (см. главы 24, 25).

3. **Командный процессор - COMMAND.COM.** файлы, **перечисленные** в пунктах (2) и (3), расположенные в определенных областях памяти, а также процедуры и данные, расположенные в ПЗУ (BIOS), в совокупности составляют операционную систему MS DOS, которая обслуживает **Вас** во время работы на компьютере.

Высокий уровень программирования (прерывания **21H,** 25H, 26H и др.) берет на себя MSDOS.SYS. Особенно полно здесь представлены процедуры работы с файловой системой. При работе с файлами редко когда появляется необходимость использовать средний (прерывание **13H)** или низкий уровень (практически никогда).

Файл IO.SYS существенно дополняет систему BIOS, расположенную в ПЗУ. Он выполняет следующие три основные задачи:

1. Настройка на **нужды** MS DOS. Оказывается, на базе BIOS работают и другие операционные системы, и все они подстраивают BIOS для своих нужд.
2. Исправление ошибок, которые могут оказаться **в ПЗУ. Для** того чтобы исправить ПЗУ, требуется больше времени, чем внести изменение в файл,
3. Управление новыми устройствами, не вошедшими в перечень устройств, обслуживание которых обеспечивает **ПЗУ** [15].

Одной из функций файла **COMMAND.COM** является интерпретация и выполнение команд, которые вводятся в командной строке.

П.

Различные уровни программирования разберем на примере вывода информации на экран. Отмечу сразу, что большая часть материала будет относиться к VGA адаптеру. В следующей главе будет рассказано, как программно определить вид вашей видеосистемы.

Высокий уровень. Обратимся в справочнике по MS DOS (Приложение **7)** на функции с номерами с 1 **по** ОСН. **Все** эти функции отвечают за работу со стандартными устройствами ввода-вывода. С двумя из функций вывода на экран **Вы** уже познакомились - это функции с номерами **02 и** 09. Они отвечают за вывод символа и вывод строки. Ранее было указано, **что не все** коды интерпретируются ими как символы. Чтобы вывести на экран символы, соответствующие всем кодам, мы использовали прерывание **BIOS** - ЮН. Функция 06Н DOS, способная как выводить, так и вводить символы, также не может вывести на экран символы управляющих кодов. Однако **это** не является большим недостатком, **так** как необходимость выводить такие символы появляется довольно редко.

Возможности MS DOS, однако, не исчерпываются специализированными функциями по выводу на экран. В операционной системе **MS** DOS реализован подход "описателей **файла".** При открытии или создании файла, DOS возвращает в регистр АХ описатель файла или HANDLE -двухбайтовое число. Все операции сданным файлом после этого можно производить, зная только это число (см. главу 8). Удобство такого подхода заключается **в** том, что стандартным устройствам по умолчанию также присваиваются свои описатели - от 0 до 4. Вот эти описатели:

---

[15] Обслуживание новых **устройств** можно **обеспечить за** счет загружаемых или резидентных драйверов.

стандартное устройство ввода (клавиатура) - O,
стандартное устройство вывода (экран) - 1,
устройство для вывода ошибок (обычно экран) - 2,
асинхронный порт(COM1) - 3,
печатающее устройство (LPT1) - 4.

Это позволяет легко перенаправить ввод или вывод с одного устройства на другое. Подробности работы с файлами оставим до главы 8, здесь же разберем следующий пример. Функция DOS 40H осуществляет запись на диск. При этом в BX должен находиться описатель файла, в CX — число записываемых байтов, а DS:DX должно указывать на буфер. Следующая программа будет выводить текстовый файл PRIMER.TXT на экран (предварительно сформируйте его с помощью текстового редактора в текущем каталоге).

```
DSEG    SEGMENT
BUFER  DB  200  DUP(?)        ;буфер ввода-вывода
FILE    DB  'PRIMER.TXT',0   ;файл в текущем каталоге
HANDLE DW ?                   ;здесь храним описатель дискового файла
EOF     DB 0                  ;если 1, то достигнут конец файла
DSEG    ENDS
SSEG SEGMENT STACK
     DB 30 DUP(?)
TOP  DB ?
SSEG ENDS
CODE SEGMENT
     ASSUME CS:CODE, DS:DSEG, SS:SSEG
BEGIN:
;готовим регистры
     MOV AX,DSEG
     MOV DS,AX
     MOV AX,SSEG
     MOV SS,AX
     MOV SP,OFFSET SS:TOP
;-открываем файл
     MOV AH,3DH
     MOV AL,0
     LEA DX,FILE
     INT 21H          ;вызов функции открытия файла
     JC  ER           ;если ошибка-закончим
     LEA DX,BUFER     ; DX на буфер ввода-вывода
     MOV BX,AX
     MOV HANDLE,BX    ;сохраним описатель
     MOV CX,200       ;читаем-пишем по 200 байт
;--
CIKL:
```

```
;читаем в буфер
      MOV AH,3FH
      INT 21H
      JC  CLOSE          ;если ошибка - закончить
      CMP CX,AX
      JZ  NOT_EOF
;конец файла достигнут
      MOV EOF,1
      MOV CX,AX
      NOT_EOF:
;пишем на экран из буфера
      MOV AH,40H
      MOV BX,1           ;описатель для вывода на экран
      INT 21H
      CMP EOF,1          ;не пора ли заканчивать
      MOV BX,HANDLE
      JZ  CLOSE
      JMP CIKL           ;продолжаем читать
;--
CLOSE:
;закрыть файл
      MOV AH,3EH
      INT 21H
ER:
;выйти в операционную систему
      MOV AH,4CH
      INT 21H
CODE ENDS
      END BEGIN
```

Рис. 6.1. Программа вывода текстового файла па экран.

Детальное обсуждение работы с файлами мы отложим до главы 8. Здесь же постарайтесь разобраться с тем, как программа работает с описателем файла. Хочу заметить, что вывод информации на экран с помощью функций DOS полностью идентичен для любого типа экрана или адаптера. Функции DOS позволяют практически отрешиться от свойств аппаратуры, чего нельзя сказать, например, о средствах BIOS. Зато средства DOS для символьного вывода слишком бедны. Выбирайте.

Вывод информации на экран на среднем уровне осуществляется посредством обращения к функциям прерывания ЮН, которое представляет довольно мощный экранный сервис для текстового режима. Мы уже выводили символы с помощью функции ОАН этого прерывания. Познакомимся и с некоторыми другими возможностями. Особенно интересны, на мой взгляд, функции 06 и 07. Они позволяют сдвигать экран или часть экрана как целое вверх или вниз на заданное число строк. Кстати, при выво-

де текста с помощью функций DOS происходит автоматическая прокрутка (сдвиг) экрана вверх - для этого как раз используется функция 06 прерывания ЮН. Здесь приводится процедура очистки экрана, которая также может быть осуществлена с помощью данной функции.

```
PROC  CLS
      MOV CX,0        ;координаты левого верхнего угла CH-y,CL-x
      MOV DX,184FH ;координаты правого нижнего угла DH-y,DL-x
      MOV AL,0        ; количество сдвигаемых строк, если 0, то весь экран
      MOV AH,06H    ;номер функции
      MOV BH,7       ;атрибут цвета
      INT 10H         ;вызов функции
      RET              ;возврат в основную программу
CLS   ENDP
```

Рис. 6.2. Процедура очистки экрана.

Процедура, приведенная на Рис.6.2, является вполне законченной, и Вы можете ее использовать в своей программе. Обращаю Ваше внимание на содержимое регистра ВН. После очистки экрана текст будет обладать заданным цветом. Номер 7 соответствует светло-серому цвету.

Следующая программа использует функцию 13Н прерывания 10Н для вывода строки. Данная функция позволяет выводить строку (подфункции 2,3) в формате: симв.,-атр., ... Работает она только для EGA или VGA адаптеров.

```
DSEG    SEGMENT
STROKA  DB 'П',1,'Р',2,'И',3,'В',4,'Е',5,'Т',6
DSEG    ENDS
SSEG SEGMENT STACK
      DB 60 DUP(?)
TOP   DB ?
SSEG ENDS
CODE  SEGMENT
      ASSUME CS:CODE, DS:DSEG, SS:SSEG
BEGIN:
      MOV AX,DSEG
      MOV DS,AX
      MOV AX,SSEG
      MOV SS,AX
      MOV SP,OFFSET SS:TOP
;---
      PUSH DS
      POP ES                          ;ES на сегмент данных
      MOV BP,OFFSET DS:STROKA         ;BP на строку
```

```
    MOV DX,0205H                 ;координаты 2,5
    MOVAL,3                      ;подфункция 3
    MOVAH,13H                    ;функция вывода строки
    MOVCX,6                      /шесть символов (только!)
    INT 10H                      /вызов прерывания
;---
    MOV AH,4CH
    INT 21H
CODE ENDS
    END BEGIN
```

Рис. 6.3. Программа вывода строки разноцветными буквами.

Обращаю внимание на то, что подпрограммы BIOS используют стек программы, поэтому если Вы делаете EXE-программу, то резервируйте больше места для стека. Обратимся, наконец, к непосредственной работе с экраном. Экранная память для текстового режима располагается, начиная с OB800H:OOOOH (для CGA, EGA, VGA адаптеров). Каждому знакоместу на экране соответствует слово. В младшем байте содержится код символа, а в старшем - цвет (цвет символа, цвет фона, признак мерцания). Например, следующий фрагмент печатает букву А в нулевой строке экрана и девятом столбце.

```
    .
    MOV AX,0B800H
    MOV ES,AX
    MOV BYTE PTR ES:[9],'A' ;символ
    MOV BYTE PTR ES:[10],5  ;цвет 5
    .
```

Первые четыре бита байта цвета дают цвет символа. Таким образом, всего возможно 16 цветов. Цвета с номерами от 0 до 7 считаются тусклыми цветами, а с номерами от 8 до 15 - яркими. Цвет фона определяется 4, 5, 6 битами. Таким образом, цвет фона может быть только тусклым. Если бит 7 равен 1, то символ мигает. Вот, в общем, то и все. Уже теперь Вы сможете производить всевозможные манипуляции с текстовым экраном. Объем видеопамяти для текстового режима составляет 32 К. Отсюда следует, что в ней можно разместить 8 экранных страниц. Многие программы используют эти страницы для своей цели. Кому не понятно, как я подсчитал количество страниц, подскажу, что на экране 25 строк и 80 столбцов, а на каждый символ приходится 2 байта. Читателю, наверное, пришла в голову мысль о тени, которой модно теперь сопровождать всякий вывод информационного окна на экран. Принцип теневого заполнения экрана довольно прост: проверяется атрибут; если цвет тусклый, то засылается 0 (черный цвет и букв и фона), если цвет яркий, то сбрасывается бит 3, а также все старшие биты. Рассмотрим теперь процедуру копирования одной видеостраницы на другую.

```
COPY_P PROC
;сохраняем все регистры, так чтобы вызов процедуры мог производиться
; из любого места программы при входе в процедуру DH содержит номер
;страницы куда копировать, а DL номер страницы, откуда копировать
      PUSH DS
      PUSH ES
      PUSH AX
      PUSH BX
      PUSH CX
      PUSH DI
      PUSH SI
      PUSHF
;----------
      PUSH DX              ;DX нам еще понадобится
      XOR  BH,BH           ;обнуляем BH
      MOV  BL,DL           ;номер страницы в BX
      MOV  AX,4000         ;будем умножать на 4000
      MUL  BX              ;умножаем
      MOV  SI,AX           ;смещение страницы источника в SI
      POP  DX              ;восстанавливаем DX
      XOR  BH,BH           ;обнуляем BH
      MOV  BL,DH           ;номер страницы получателя в BX
      MOV  AX,4000         ;будем умножать на 4000
      MUL  BX              ;умножаем
      MOV  DI,AX           ;смещение страницы получателя в DI
      MOV  AX,0B800H       ;сегментный адрес видеобуфера
      MOV  ES,AX           ;в ES
      MOV  DS,AX           ;в DS
      MOV  CX,2000         ;длина страницы 2000 слов
      CLD                  ;флаг направления
REP MOVSW                  ;копируем
;--------------
; восстанавливаем регистры и выходим
      POPF
      POP SI
      POP DI
      POP CX
      POP BX
      POP AX
      POP ES
      POP DS
      RET
COPY_P ENDP
```

Рис. 6.4. Процедура копирования одной видеостраницы на другую.

Предложенная процедура очень удобна. Вы можете использовать ее не только в своих программах на ассемблере, но и на языках высокого уровня. Правда, требуется некоторая доработка. Проблема заключается в том, что данная процедура требует параметра. Предполагается, что этот параметр будет передаваться через регистр DX. Однако в языках высокого уровня принято передавать параметр через стек. Как видоизменить процедуру, чтобы она начала работать и для языков высокого уровня, Вы узнаете в главе 15.

III.

Здесь кратко перечислены стандартная аппаратура и средства работы с ней на всех трех уровнях.

<Принтер.>
Высокий уровень - функция DOS 05 или метод описателей
с предопределенным номером 4.
Средний уровень - прерывание BIOS 17H. ($)
Низкий уровень - порты ввода-вывода параллельного адаптера.

<Клавиатура.>
Высокий уровень - ввод с помощью стандартных
функций DOS или с помощью описателя файла 0 ($).
Средний уровень - прерывание BIOS 16H, буфер клавиатуры ($).
Низкий уровень - порты клавиатуры с перехватом прерывания 9H.

<Последовательныйпорт.>
Высокий уровень - функция DOS 04 или вывод с помощью описателя 03.
Средний уровень - использование функций прерывания 14H.
Низкий уровень - обращение непосредственно к портам адаптера.

<Дисковые накопители.>
Высокий уровень - богатый выбор функций DOS,
а также прерывания 25H и 26H. ($)
Средний уровень - прерывание 13H - абсолютная запись на диск.
Низкий уровень - порты контролера диска. (@)

<Дисплей, графические режимы.>
Высокий уровень - нет средств.
Средний уровень - функции прерывания ЮH.
Низкий уровень - порты адаптера дисплея. ($)

Значок $ означает - употребляется чаще всего.
Значок @ означает - почти не употребляется.

IV. Программирование звука.

Что за звуки! Неподвижен, внемлю
Сладким звукам я...

М.Ю. Лермонтов.

В DOS и BIOS отсутствуют какие-либо средства, позволяющие извлекать звук из динамика компьютера, кроме разве что посылки кода 7 через стандартные функции вывода (MOV DL,7/MOV AH,2/INT 21H). В основе генерации звука лежит взаимодействие микросхемы таймера и динамика. Микросхема таймера считает импульсы, получаемые от тактового генератора, и может по прошествию определенного количества импульсов (это можно запрограммировать) выдавать на выход сигнал. Если эти сигналы направить на вход динамика, то будет произведен звук. Высота его будет зависеть от частоты поступления сигналов на вход динамика. Вот кратко идея генерации звука в стандартной конфигурации IBM PC. А сейчас об этом более подробно (описание таймера см. в Приложении 9).

Микросхема таймера имеет три канала. Канал 0 отвечает за ход системных часов. Сигнал с этого канала вызывает прерывание времени. 18.2 раз в секунду выполняется процедура, на которую направлен вектор с номером 8. Это процедура производит изменения в области памяти, где хранится текущее время. В специальном регистре задвижки хранится число синхроимпульсов, по прошествии которых сигнал таймера должен вызвать прерывание времени. Уменьшая это число (через порт канала), можно заставить идти системные часы быстрее. Адрес порта канала 0 - 40H.

Канал 1 отвечает за регенерацию памяти. Адрес порта этого канала - 41H. В принципе можно уменьшить число циклов регенерации памяти в секунду, что может несколько увеличить производительность компьютера. Однако это можно сделать лишь в некоторых пределах, т.к. при увеличении промежутка регенерации возрастает вероятность сбоя памяти.

Канал 2 обычно используется для работы с динамиком, хотя сигналы с него можно использовать и для других целей. Адрес порта этого канала - 42H. Идея генерации звука проста. В порт 42H посылается число (счетчик). Немедленно значение счетчика начинает уменьшаться. По достижению 0 на динамик подается сигнал. После чего процесс повторяется. Чем больше значение счетчика, тем реже подается сигнал и тем ниже звук.

Связь канала 2 с динамиком устанавливается через порт 61H. Если бит 1 этого порта установлен в 1, то канал 2 посылает сигналы на динамик. Кроме того, чтобы разрешить поступления сигнала от тактового генератора в канал 2 бит 0 этого порта должен быть равен 1 (уровень сигнала высокий).

Для программирования каналов сначала требуется установить порте адресом 43H. Значения битов этого порта представлены ниже.

Бит	Значение
O	0 - двоичные данные, 1 - данные в двоично-десятичном виде;
1-3	номер режима, обычно используется режим 3;
4-5	тип операции: 00 передать значение счетчика в задвижку, 01 читать/писать только старший байт, 10 читать/ писать только младший байт, 11 читать/писать старший байт, потом младший;
6-7	номер программируемого канала (0-2).

```
CODE  SEGMENT
      ORG 100H
      ASSUME CS:CODE
BEGIN:
      MOV  AL,10110110B                  ;установка режима записи
      OUT  43H,AL
      IN   AL,61H
      OR   AL,3                          ;разрешить связь с таймером
      OUT  61H,AL
      MOV  AX,1200
;установить частоту звука
;таймер начинает действовать немедленно по
;засылке счетчика
      OUT  42H,AL
      MOV  AL,AH
      OUT  42H,AL
      MOV  CX,OFFFFH
;задержка
      LOO:  LOOP LOO
;отключить канал от динамика, т.е. прекратить звук
      IN   AL,61H
      AND  AL,11111100B
      OUT  61H,AL
      RET
CODE  ENDS
      END BEGIN
```

Рис. 6.5. Программа, производящая короткий звуковой сигнал.

В программе, представленной на Рис. 6.5, дан механизм создания звуковых эффектов. Вы можете использовать его в своих программах. Есть два основных подхода создания в программе для MS DOS звукового (музыкального) оформления. Первый

подход заключается в циклическом обращении к соответствующей процедуре. В этом случае фактически Программа не сможет выполнять другую работу. Другой способ позволяет создавать музыкальный фон, программа при этом может заниматься своими делами. При этом производительность ее снижается очень незначительно. Суть идеи заключается в использовании прерывания по времени. Соответствующий вектор направляется на процедуру, и к ней периодически происходит обращение уже независимо от того, что выполняет программа. О том, как это сделать, Вы узнаете в следующих главах.

Как уже говорилось, микросхема таймера служит для получения системой информации о текущем времени и дате, которые можно получить или установить с помощью функций прерывания 21H: 2AH, 2BH, 2CH, 2DH. В свою очередь, прерывание BIOS 1AH позволяет получить количество тиков системных часов, прошедших с момента установки. Один тик=(1/18.2). Во многих случаях удобнее пользоваться именно этим прерыванием, а не функциями MS DOS. На компьютерах класса AT есть встроенные часы реального времени, которые идут независимо от системных и при отключенном питании. Доступ к ним можно осуществить также через прерывание 1AH. После запуска компьютера системные часы устанавливаются по часам реального времени. В дальнейшем они идут независимо. Программная модель часов реального времени изложена в Приложении 9.

Глава 7. Клавиатура, дисплей, принтер.

Для выхода в меню нажмите клавишу "Reset".
Где находится клавиша "Any key"?
Семеро одного дисплея не ждут.

Программисты шутят.

Данная глава является продолжением предыдущей. На примере клавиатуры, дисплея, принтера мы рассмотрим способы управления внешними устройствами. По тому, как программа справляется с ними, судят о мастерстве и профессионализме автора.

I.

Рассмотрим цепочку событий, которые происходят после нажатия некоторой клавиши. Мы анализируем ситуацию как бы с точки зрения микропроцессора. Более подробно об управлении клавиатурой см. Приложение 9.

После нажатия клавиши на микропроцессор, на вход INT, поступает сигнал прерывания от контролера прерываний (см. ниже). После этого микропроцессор заканчивает выполнение текущей команды и получает с шины номер прерывания, в данном случае 9. Затем микропроцессор выполняет команду INT 9H [16]. После выполнения процедуры прерывания микропроцессор начинает выполнять следующую команду. Процедура, на которую показывает вектор 9H, считывает из портов клавиатуры скан-код нажатой клавиши - каждой клавише присваивается свой скан-код (не путать с ASCII кодом). Данный скан-код анализируется на предмет того, какая клавиша нажата - алфавитно-цифровая, расширенная (с расширенным кодом ASCII) или управляющая клавиша. Если клавиша алфавитно-цифровая, то ее ASCII код и скан-код помещаются в буфер клавиатуры (см. ниже). Если клавиша имеет расширенный код, то этот код также помещается в буфер клавиатуры вместе с нулевым байтом [17]. Наконец, если нажата управляющая клавиша, то 9-е прерывание меняет соответствующий флаг в слове состояния клавиатуры, расположенном по адресу 0040H:0017H. Таким образом, одни клавиши изменяют содержимое буфера клавиатуры, а другие - слово состояния клавиатуры. Единственным исключением остается клавиша Ins. Она имеет расширенный код ASCII, и в тоже время данные о нажатии этой клавиши заносятся в слово-состояние клавиатуры. Кроме этого, 9-е прерывание выполняет ряд специальных функций: распознает нажатие клавиш PrtSc, Ctrl-Break и производит соответствующие действия. На этом миссия этого прерывания заканчивается. Значение битов слова-состояния клавиатуры показано на Рис. 7.1. На современной клавиатуре появился (и появляется) ряд новых клавиш. Они в основном предназначены для операционной сис-

---

[16] О прерываниях смотри главу 9.

[17] Занимаясь программированием на любом языке, Вы, несомненно, должны знать, что расширенный код ASCII представляет собой двухбайтную величину, первый байт которой равен нулю.

темы Windows. Операционная система MS DOS не обрабатывает эти клавиши. Однако прерывание при нажатии этих клавиш происходит, и Вы (по прочтению главы 9) сможете сами обрабатывать эти клавиши.

40H:17H →

0: 1 - правая Shift нажата;
1: 1 - левая Shift нажата;
2: 1 - Ctrl (любая) нажата;
3: 1 - Alt (любая) нажата;
4: 1 - режим ScrollLock;
5: 1 - режим NumLock;
6: 1 - режим CapsLock;
7: 1 - режим Insert;

40H:18H →

0: 1 - левая Ctrl нажата;
1: 1 - левая Alt нажата;
2: 1 - SysRec нажата;
3: 1 - пауза;
4: 1 - ScrollLock нажата;
5: 1 - NumLock нажата;
6: 1 - CapsLock нажата;
7: 1 - Insert нажата;

Рис. 7.1. Слово-состояние клавиатуры.

Слово-состояние клавиатуры и буфер клавиатуры являются отправной точкой для работы специальной процедуры BIOS - прерывания 16H (см. Приложение 8). Мы с ним уже встречались. Функции DOS получают сведения о нажатых клавишах только через это прерывание. Мы познакомились только с функцией 0 данного прерывания. Но есть и другие полезные функции.

Функция 1 - получить последний введенный символ из буфера клавиатуры, не меняя его содержимое (функция 0 удаляет символ из буфера) и не ожидая нажатия.

Функция 2 - дает первый байт **слова-состояния** клавиатуры, **функция** 5 вставляет символ в буфер клавиатуры **и** т.д.

Следующая программа - пример управления курсором на текстовом экране. Курсор управляется обычными клавишами управления. По нажатию клавиши **ESC** программа заканчивает свою работу. Границами перемещения курсора являются границы экрана.

```
CODE  SEGMENT
      ASSUME CS:CODE
      ORG 100H
BEGIN:
;определяем текущее положение курсора
      MOV BH,0
      MOV AH,03
      INT ЮН
;текущее положение теперь в DH-строка, DL-столбец
LOO:
;ждем нажатия клавиши
      MOV AH,0
      INT 16H
;если ESC, то выход
      CMP AL,27
      JZ  EXIT
      CMP AL,0
      JNZ LOO      ;если не расширенный код, то повторяем ввод
      CMP AH,50H /курсор вниз?
      JNZ N01
      CMP DH,24       ;граница?
      JZ  LOO
      INC DH
      JMP SHORT EX
N01:
      CMP AH,48H      ;курсор вверх?
      JNZ N02
      CMP DH,0        ;граница?
      JZ  LOO
      DEC DH
      JMP SHORT EX
N02:
      CMP AH,4BH      ;курсор влево?
      JNZ NO3
      CMP DL,0        ;граница?
      JZ  LOO
      DEC DL
      JMP SHORT EX
```

```
NO3:
     CMP AH,4DH      /курсор вправо?
     JNZ LOO
     CMP DL,79       ;граница?
     JZ  LOO
     INC DL
EX:
;устанавливаем новое положение курсора
;новые координаты в DH,DL
     XOR BH,BH       ;страница 0
     MOV AH,02
     INT 10H          /ставим курсор
     JMP LOO
EXIT:
     MOV AH,4CH
     INT 21H
CODE ENDS
     END BEGIN
```

Рис. 7.2. Программа управления курсором.

Обратимся теперь к буферу клавиатуры. Буфер клавиатуры располагается в сегменте 40Н. Он имеет кольцевую структуру. Смещение головы его хранится в байте с адресом 1АН (сегмент тот же). Смещение хвоста — в байте с адресом 1СН. Для самого буфера отведено пространство с 30Н по 60Н. Для записи клавиш отведено два байта: младший - код ASCII (или 0, если расширенный код), старший - скан-код (или второй байт расширенного кода). Таким образом, в буфере может храниться пятнадцать нажатий клавиш. Буфер становится переполненным, когда разность между содержимым байта 1АН и байта 1СН становится равным двум (содержимое 1СН на 2 меньше содержимого ячейки 1АН) либо в частном случае в 1АН будет 30, а в 1СН - 60. Алгоритм того, как вставить очередной символ в буфер клавиатуры, можно описать следующими словами:

1. Проверяем, не переполнен ли буфер. Если не переполнен, то переходим к пункту 2, в противном случае даем звуковой сигнал.

2. Помещаем код клавиши в ячейку, на которую указывает 1СН. Скан-код помещаем в следующий байт. Если клавиша имеет расширенный код, то в первую ячейку поместим 0, а во вторую ее расширенный код.

3. Если содержимое 1СН было равно 60, то засылаем туда 30, в противном случае увеличиваем содержимое на 2.

Если Вы уже достаточно сильны в ассемблере, попробуйте реализовать этот алгоритм [18]. Кстати, для очистки буфера клавиатуры нужно, чтобы и 1АН и 1СН содержали одинаковое значение. Ниже представлена процедура очистки буфера клавиатуры.

---

[18] **Данный алгоритм реализован в прерывании 9Н. Вряд ли Вам понадобится писать свое собственное прерывание обработки клавиатуры. О том, как перехватить готовое прерывание и воспользоваться им, мы поговорим в главе 9.**

```
CLRBUF   PROC
     CLI
     PUSH  AX
     PUSH  ES
     MOV   AX,40H
     MOV   ES,AX
     MOV   AL,BYTE PTR ES:[1CH]
     MOV   BYTE PTR ES:[1AH],AL
     POP   ES
     POP   AX
     STI
     RET
CLRBUF   ENDP
```

Рис. 7.3. Очистка буфера клавиатуры.

На Рис. 7.4 показана процедура вставки в буфер клавиатуры кода символа. Код ASCII находится в CL, скан-код — в CH. В процедуре реализован алгоритм, описанный выше словесно.

```
IN_BUF  PROC
     CLI
     PUSH  ES
     PUSH  SI
     PUSH  DI
     MOV   DI,40H
     MOV   ES,DI
     MOV   DI,ES:[1CH]  ;хвост
     MOV   SI,ES:[1AH]  ;голова
     CMP   DI,60
     JZ    SPEC
     ADD   DI,2
     CMP   DI,SI
     JZ    FUL              ;буфер переполнен?
     MOV   ES:[DI-2],CL
     MOV   ES:[DI-1],CH
     JMP   SHORT EN
SPEC:                          ;специальный случай
     CMP   SI,30
     JZ    FUL              ;буфер переполнен
     MOV   ES:[DI],CL
     MOV   ES:[DI+1],CH
     MOV   DI,30
EN:
     MOV   ES:[1CH],DI
```

```
FUL:
        POP   DI
        POP   SI
        POP   ES
        STI
        RETN
IN_BUF  ENDP
```

Рис. 7.4. Процедура, вставляющая код символа в буфер клавиатуры.

Теперь Вы знаете достаточно, чтобы программно манипулировать буфером. Однако помните, что на время работы с ним желательно отключать прерывания (CLI) - нажатие клавиши может быть сделано в момент работы программы с буфером. В этом случае могут произойти непредсказуемые события. В главе 17 будет приведен пример использования буфера клавиатуры.

Рассмотрим следующий простой пример (Рис. 7.5). Выход из программы происходит лишь при одновременном нажатии клавиш CTRL, ALT, F1.

```
CODE   SEGMENT
ASSUME  CS:CODE
        ORG  100H
BEGIN:
;ждем нажатия  клавиши
        MOV AH,0
        INT  16H
        CMP AL,0
        JNZ  BEGIN       ;если код не расширенный - повторить
        CMP  AH,94       ;код одновременного нажатия Ctrl и F1
        JZ   PROV
        CMP  AH,104      ;код одновременного нажатия Alt и F1
        JZ   PROV
        JMP  SHORT BEGIN
PROV:
;здесь проверяем биты состояний клавиш Ctrl  и Alt
        MOV  AX,40H
        MOV  ES,AX
        TEST BYTE PTR ES:[17H],00000100B ;проверка слова состояния
;  клавиатуры на клавишу Ctrl
        JZ   BEGIN
        TEST BYTE PTR ES:[17H],00001000B ;проверка слова состояния
;клавиатуры на клавишу Alt
        JZ    BEGIN
EXIT:
        MOV AH,4CH
        INT  21H
CODE  ENDS
        END BEGIN
```

Рис. 7.5. Выход из программы происходит при одновременном нажатии CTRL, ALT, Fl.

Идея этой программы состоит в том, что сочетания Ctrl F1 и Alt F1 имеют свои расширенные коды. Когда же нажимают все три клавиши, то неизвестно, какой код реализован на самом деле. Мы проверяем и тот, и другой случай, тем самым убеждаясь, что по крайней мере нажата клавиша F1. А затем проверяем флаги и выясняем, нажаты ли одновременно клавиши Ctrl и Alt. Если перехватывать прерывание 9H, то ту же процедуру можно сделать проще. Действительно, в этом случае нужно проверить только скан-код клавиши F1, а затем статус управляющих клавиш.

А сейчас мы рассмотрим вопрос, который волнует многих и, который до конца не знают даже некоторые опытные программисты. Вопрос собственно состоит в том, что значит для приложений MS DOS Ctrl C и что значит Ctrl Break и "как с ними бороться"?

Первое, что хотелось бы сказать, это то, что ситуации, когда эти прерывания необходимо исключить, существуют. В конце концов пользователь может случайно или по инерции нажать нежелательное сочетание клавиш, и прерывание программы произойдет в самый нежелательный момент. Я уже не говорю о том, что появление значка «^С» может испортить эстетическое восприятие программы пользователем.

Чувствительными к нажатию известного сочетания клавиш являются только функции DOS, процедуры BIOS не обращают на это внимание. Уровень чувствительности функций DOS можно регулировать. Есть два таких уровня:

1. Чувствительны к Ctrl Break (Ctrl C) только функции символьного ввода-вывода - экран, принтер, в параллельный порт.
2. Чувствительна большая часть функций DOS.

Чувствительность можно установить в файле CONFIG.SYS либо прямо в командной строке командами BREAK OFF (первый уровень чувствительности) и BREAK ON (второй уровень чувствительности). Узнать, какой уровень чувствительности установлен в системе, можно из командной строки командой BREAK. Из программы это можно сделать, используя функцию DOS за номером 33H, с помощью этой функции можно также изменить уровень чувствительности. Предположим, что в Вашей программе не используются функции DOS символьного ввода-вывода. В этом случае, для того чтобы обезопасить себя от ненужного прерывания, достаточно установить первый уровень чувствительности[19]. Правда, при выходе из программы может появиться значок «^С», но это уже мелочи. Изложенному подходу присвоим номер 1.

При распознавании Ctrl Break или Ctrl C функции DOS выполняют команду INT 23H. 23-е прерывание осуществляет при этом выход из программы в порождающий процесс (обычно таковым является просто MS DOS). Второй способ устранения неприятного прерывания заключается в том, что Вы перехватываете прерывание 23H (см. главу 9) и направляете на свою процедуру, в которой может стоять только одна команда - IRET. Таким образом Вы заблокируете выход по Ctrl Break (Ctrl C). Однако на экране по-прежнему может появляться значок «^С».

Стоит поговорить подробнее о том, что происходит при нажатии клавиш Ctrl Break. Распознавание этих клавиш происходит еще на уровне 9-го прерывания. При этом

---

[19] После выполнения программы следует восстановить исходные установки - правила хорошего тона.

выполняется прерывание 1BH. В процедуре, которая при этом выполняется, стоят всего две команды:

```
MOV BYTE PTR CS:[MEM],3
IRET
```

где MEM - некоторая ячейка памяти. Наличие в этой ячейке числа 3 является индикатором того, что было нажато сочетание клавиш Ctrl Break. При нажатии такого сочетания клавиш ко всему прочему очищается буфер клавиатуры. При обнаружении в MEM числа 3 MS DOS немедленно выполняет команду INT 23H. В случае с Ctrl C ситуация иная. Это сочетание распознается на уровне MS DOS. Дело в том, что у этого сочетания есть свой код - это 3. Появление данного кода в буфере клавиатуры распознается MS DOS, и опять выполняется команда INT 23H. Вы можете сэмитировать нажатие клавиш Ctrl Break, послав в ячейку MEM число 3, а нажатие клавиш Ctrl C — послав в буфер клавиатуры также число 3 (второй байт - скан-код может быть любым).

Из всего сказанного вытекает еще один способ устранения нежелательного прерывания. Вектор 1BH направляется на процедуру, где стоит всего одна команда IRET. Еще остается Ctrl C. Попадание кода 3 в буфер клавиатуры можно блокировать либо на уровне 16-го,либо на уровне 9-го прерывания. Например, на уровне 9-го прерывания Вы определяете, не нажата ли клавиша C, и если нажата, то проверяете статус клавиши Ctrl. Если она также нажата, то Вы сбрасываете соответствующий бит в слове состояния клавиатуры. Двух этих операций достаточно, чтобы не обращать никакого внимания ни на статус Ctrl Break, ни на то, какие функции MS DOS выполняет программа (см. конец главы 9).

П.

В предыдущей главе мы довольно долго говорили о работе с текстовым экраном. Отдавая себе отчет, что объем материала здесь просто неограничен, я все же изложу здесь (далеко не полно, только для возбуждения интереса) лишь один вопрос - загружаемые символы. Вы, надеюсь, знаете, что импортная техника не русифицирована, т.е. для того, чтобы писать и читать русские тексты, необходимо вначале загрузить специальный драйвер. В специальной области памяти хранятся шаблоны символов. Эта область памяти называется знакогенератором: речь идет о EGA или VGA-адаптере. При инициализации системы BIOS загружает в знакогенератор шаблоны символов. Символы с кодами от 0 до 127 - это стандарт. Часть же символов с кодами от 128 до 255 можно использовать для введения национальных алфавитов. В частности — русского алфавита. Этим и занимаются всевозможные русифицирующие драйверы. Стандартные шаблоны символов находятся в ПЗУ, на начало этих шаблонов показывает вектор 44H. При выполнении инициализации режима экрана символы из ПЗУ копируются в знакогенератор.

Перед таким драйвером экрана-клавиатуры [20] стоят четыре задачи. Первая заключается в том, чтобы загрузить в знакогенератор шаблоны символов. Вторая, более труд-

[20] Вообще говоря, русификация дисплея и клавиатуры - две разные задачи, решения которых совсем не обязательно объединять в один драйвер.

ная, состоит в том, чтобы установить переключатели для перехода от русского шрифта к латинскому и обратно. Обычно это различные клавиши-переключатели. Задача решается перехватом прерывания клавиатуры и расшифровки того, нажата клавиша в русском или латинском регистре. После расшифровки клавише будет присвоен тот или иной код ASCII. Третья задача состоит в поддержке русского алфавита в графических режимах. Мы не будем это рассматривать. Замечу только, что часто символы с кодами выше 127 в графическом режиме вообще никак не представлены (в отличие от текстового режима). Наконец четвертая задача драйвера заключается в том, чтобы не дать другим программам испортить загруженные в знакогенератор шаблоны (а испортить можно очень просто, если вызвать функцию 0 прерывания ЮН установка режима экрана). Для этого приходится отслеживать все вызовы прерывания 1ОН.

Не вдаваясь в подробности, рассмотрим только, как можно загрузить в знакогенератор свои шаблоны символов. Вы можете использовать это в своей программе, если захотите работать с нестандартными символами. При выходе из нее легко восстановить прежнее состояние, выполнив команды:

```
MOV AX,0003H
INT  10H
```

При этом знакогенератор загружается символами из ПЗУ. Чтобы не портить содержимое экрана, добавьте к содержимому AL 80H.

В качестве иллюстрации к сказанному на Рис. 7.5 представлена простая программа, после выполнения которой символ буквы А (код 65) будет заменен на прямоугольник. Программа иллюстрирует работу подфункции 0 функции 11Н прерывания 1ОН. Функция 11Н прерывания 1ОН - довольно сложная штука, и Вам не миновать ее, если требуется манипулировать экранными шрифтами.

```
CODE  SEGMENT
      ASSUME  CS:CODE
      ORG  100H
BEGIN:
      JMP  BEG
;шаблон для буквы А, точнее символа с кодом 65
FONT:
      DB  11111111B
      DB  10000001B
      DB  10000001B
      DB  10000001B
      DB  10000001B
      DB  10000001B
      DB  10000001B
      DB  10000001B
      DB  40000001B
      DB  10000001B
```

```
    DB 10000001B
    DB 10000001B
    DB 10000001B
    DB 11111111B
BEG:
;следующие две команды для COM-программы не обязательны.
;Почему?
    PUSH CS
    POP  ES
    MOV BP,OFFSET CS:FONT
;ES:BP теперь указывает на шаблон
    MOV CX,1            ;один символ
    MOV DX,65           ;буква А латинская
    MOV BL, 0           ;блок 0, отображаемый по умолчанию
    MOV BH,14           ;в шаблоне 14 строк (байт)
    MOV AL,0            ;подфункция 0
    MOV AH,11H          ;функция 11H
    INT 10H             ;вызов прерывания
    MOV AH,4CH
    INT 21H
CODE ENDS
    END BEGIN
```

Рис. 7.6. Загрузка в знакогенератор шаблона символа А (65).

Ш. О различных видеоадаптерах.

Материал данного раздела носит несколько исторический оттенок.

В нашей книге мы касаемся в основном VGA-адаптеров. Между тем вопрос о видеоадаптерах - больной вопрос для многих программистов [21]. В текстовом режиме для VGA-адаптеров имеется 32 килобайта памяти. Есть соблазн использовать эту память в своих программах. Но у CGA-адаптеров памяти в текстовом режиме 16К, а у Hercules'a всего 4К. Кроме того, видеобуфер у последнего адаптера начинается с адреса 0B000H. Поэтому если нужно, чтобы Ваша программа работала на всех адаптерах, откажитесь от использования страниц видеопамяти. Мой совет: выделите в ОЗУ некоторую область памяти и эмулируйте работу с ней, как с видеопамятью. Таким образом, по крайней мере, часть проблемы по совместимости программы Вы решите.

Другая проблема - это адрес видеобуфера. Здесь проще всего поступить следующим образом. Причем можно обойтись без тестирования видеосистемы компьютера. Проверьте режим экрана, если режим окажется равным 7, то попытайтесь переустановить его в 3-й. Если это удалось, то буфер начинается с OB800H, а если нет — то с

[21] Конечно сейчас CGA - уже экзотика. Материал, представленный здесь, носит историко-просветительский характер.

ОВОООН. Как видим, алгоритм весьма прост. Необходимость переустановки связана с тем, что VGA-адаптеры при запуске компьютера устанавливают режим 7 на монохромном дисплее и 3 на цветном.

Если Ваша программа, однако, использует некоторые другие возможности видеосистемы, например, загружает в знакогенератор свои символы, то не обойтись без более тщательного тестирования адаптера. Ниже приводится процедура, которая определяет вид адаптера. В AL возвращается 0 для VGA, 1 для EGA, 2 для CGA, 3 - MDA, 4 - Hercules (HGC).

```
WHAT_AD   PROC
       PUSH  DX
       PUSH  BX
       PUSH  CX;проверка наличия у прерывания ЮН функции 1АН - есть у VGA
       XOR  AL,AL
       MOV AH,1AH
       INT  10H
       CMP AL,1AH
       JNZ  NO_VGA
       XOR AL,AL
       JMP  SHORT  EXIT
NO_VGA:
;проверка наличия у прерывания ЮН функции 12H - есть у EGA
       MOV AH,12H
       MOV BL,10H
       INT  ЮН
       CMP BL,10H
       JZ   NO_EGA
       MOV AL,1
       JMP  SHORT  EXIT
NO_EGA:
;проверка наличия CGA
       MOV DX,3D4H
       CALL  SEARCH_6845
       JC   NO_CGA
       MOV   AL,2
       JMP   SHORT  EXIT
NO_CGA:
;проверка наличия MDA
       MOV   DX,3B4H
       CALL  SEARCH_6845
       JC    NO_MDA
       MOV   AL,3
       JMP   SHORT  EXIT
NO_MDA:
       MOV   AL,4
```

```
EXIT:
        POP   CX
        POP   BX
        POP   DX
        RET
WHAT_AD ENDP
;проверка наличия контролера 6845
;проверка осуществляется путем записи в регистр, а потом
;чтения из него если значения совпали, то контролер присутствует
;номер регистра OFH, адрес же порта у CGA и MDA различные
SEARCH_6845 PROC
        MOV AL,0FH
        OUT DX,AL
        INC DX
        IN  AL,DX
        MOV AH,AL
        MOV AL,66H
        OUT DX,AL
        MOV CX,100H
DELAY:
        LOOP DELAY
        IN AL,DX
        XCHG AH,AL
        OUT DX,AL
        CMP AH,66H
        JZ  QUIT
        STC
QUIT:
        RET
SEARCH_6845 ENDP
```

РМС. *7.7. Процедура определения видеоадаптера.*

Как уже отмечалось, материал, изложенный выше, носит исторический характер. Однако сам метод определения, точнее, алгоритм, который можно назвать "последовательным исключением", прошу взять на заметку.

VI.

Данный раздел посвящен работе принтера. Материал поистине необъятный, но он будет сужен, если условимся говорить только о средствах работы с принтером и опустим подробности работы самого принтера. Приведя довольно полный пример по выводу на печать текста, я отсылаю за подробностями управления принтером к великолепному справочнику [5].

Выше был уже рассмотрен пример, как с помощью описателей можно направить вывод на принтер. У MS DOS есть специализированная функция для вывода на печатающее устройство - номер этой функции 5. К сожалению, эта функция не дает диагностики ошибок устройства.

В BIOS есть специализированная процедура вывода на печатающее устройство. На нее направлен вектор 17H. Существенно то, что она позволяет работать с тремя принтерами (порты LPT1, LPT2, LPT3), тогда как функции DOS работают только с первым принтером - LPT1. Кроме того, данное прерывание позволяет инициализировать принтер и дает полную диагностику ошибок на этом устройстве (см. [5,13]).

Данным прерыванием пользуются многие драйверы. И в некоторых случаях желательно обойти его, т.е. обратиться непосредственно к портам адаптера. Например, вывод можно сделать через INT 17H, а проверку готовности принтера осуществить, обратившись непосредственно к портам принтера.

На Рис. 7.7 приведена программа печати текстового файла, имя которого набирается в командной строке вместе с именем программы. Программа достаточно сложная. К ней Вам еще придется вернуться после изучения работы с файлами. Поэтому я не буду объяснять строки, относящиеся к чтению файла. Разберем другие моменты.

По адресу DS:[81H] располагаются параметры, которые набирались в командной строке. В DS:[80H] лежит длина командной строки, но мы не используем этот байт. Если в командной строке ничего не набиралось, то по адресу DS:[81H] будет лежать 0DH. Если же в командной строке что-то было набрано, то необходимо учесть наличие в строке пробелов, как перед параметром, так и после него (если в строке он не один). Мы делаем это путем проверки каждого символа на пробел и фиксацией (с помощью регистра DL) того, что первая цепочка пробелов уже закончилась.

Перед каждым выводом на печать символа с помощью порта статуса определяем готовность устройства. Делается до 400 проверок, прежде чем дается сообщение о готовности принтера. Если принтер готов, то символ из буфера посылается на печать с помощью функции 0 прерывания 17H. После посылки символа проверяется байт в AH на случай ошибки вывода.

Адрес порта для первого принтера (обычно только он и есть) получаем в ячейке 40H:08H. Это так называемый базисный адрес. В нем содержится адрес порта принтера, в который посылают данные. Увеличив его на 1, мы получим порт статуса принтера. Порт управления принтером можно получить, увеличив базисное значение на 2.

```
CODE  SEGMENT
      ASSUME CS:CODE
      ORG 100H
BEGIN:
      JMP  BEG
TEXT1 DB 'Нет параметров.',13,10,'$'
```

```
TEXT2 DB 'Файл не найден.',13,10,'$'
TEXT3 DB 'Принтер не готов.Прекратить печать?(Y/N)', 13,10,'$'
PATH  DB 80 DUP(O)       /путь к файлу
BUF   DB 160 DUP{?)      ;буфер для чтения файла
PORT  DW ?               ;адрес порта статуса принтера
PRIZ  DB 0               ;признак последнего считанного блока
COUNT DW ?               /количество считанных символов
BEG:
     XOR SI,SI
     XOR DI,DI
     MOV DL, 1    /служит для определения конца первого параметра
LOO:
     CMP BYTE PTR [81H+SI],0DH
     JZ  NO_PAR
     MOV AL,[81H+SI]
     CMP AL,' '          ;пропустим пробел
     JZ  SPACE
     XOR DL,DL           ;начался первый параметр
     MOV [PATH+DI],AL    /символ в PATH
     INC DI
     JMP. SHORT L001
SPACE:
     OR  DL,DL        ;если DL=0, тогда первый параметр закончился
     JZ  NO_PAR
LOO1:
     INC SI
     JMP SHORT LOO
NO_PAR:
     OR  SI,SI           /был ли параметр
     JNZ CONT
;сообщение, затем выходим
     MOV DX,OFFSET TEXT1
     MOV AH,9
     INT 21H
     JMP EXIT
CONT:
;открываем файл
     LEA DX,PATH
     MOV AX,3D00H
     INT 21H
     JNC CONT1
/файла с таким именем, по-видимому, нет
     MOV DX,OFFSET TEXT2
     MOV AH,9
```

```
     INT  21H
     JMP  SHORT  EXIT
CONT1:
     MOV  BX,AX              ;теперь описатель будет в BX
;определяем порт статуса
;проверяем только LPT1
     MOV  AX,40H             ;адрес базового порта по адресу 40H:08H
     MOV  ES,AX
     MOV  DX,ES:[8H]
     INC  DX                 ;порт статуса на 1 больше
     MOV  PORT,DX
;читаем в буфер
PR_CONT:
     MOV  DX,OFFSET BUF      ;читаем в BUF
     MOV  CX,160             ;160 байт
     MOV  AH,3FH
     INT  21H
     CMP  AX,0
     JZ   CLOSE              ;буфер пуст, заканчиваем
     MOV  COUNT,AX           ;сколько считали - в COUNT
     CMP  AX,CX              ;не последний ли блок
     JZ   NORM
     MOV  PRIZ,1             ;если 1, то блок был последним
NORM:
     LEA  SI,BUF
N1:
     XOR  CX,CX
     MOV  DX,PORT
NORM2:
     IN   AL,DX               /читаем порт
;теперь проверяем готовность
     TEST AL,00010000B
     JZ   N2
     TEST  AL,00001000B
     JZ   N2
     TEST AL,10000000B
     JNZ   NORM1
N2:
     CMP   CX,400  /будем читать 400 раз, и только тогда сообщение
     JZ    PAUSE   /принтер не готов - сообщаем
     INC   CX
     JMP  SHORT NORM2
NORM1:
     XOR  AH,AH               ;функция 0
```

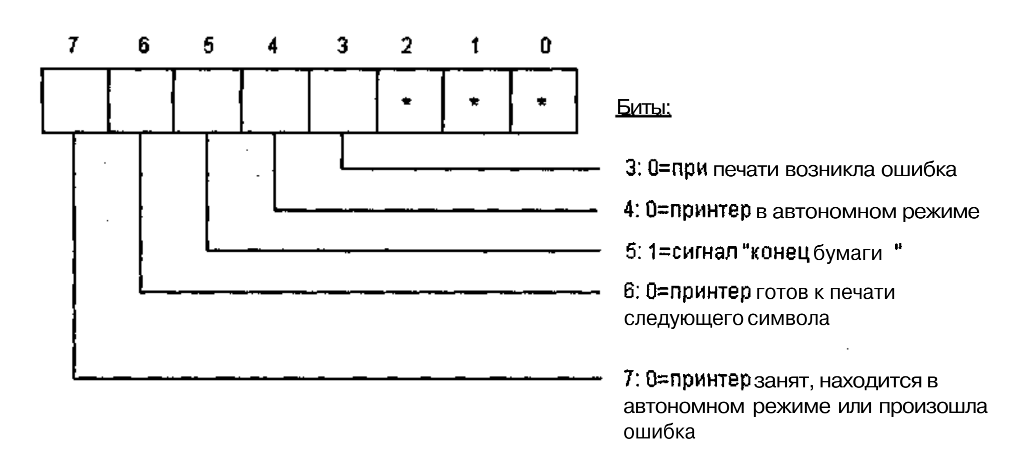
```
      XOR  DX,DX                  ;принтер LPT1
      MOV AL, [SI]                ;символ в AL
      INT 17H                     ;печатаем
      CMP AH,1                    ;проверка на time-out
      JZ   N1                    ;если ошибка,  то на проверку статуса
      INC  SI                     ;на следующий символ
      DEC COUNT                  ;уменьшаем счетчик
      JNZ N1                    ;если не равен 0,  то продолжить печатать
      CMP  PRIZ,1                 ;не кончился ли файл
      JNZ  PR_CONT              ;если не кончился,  то читать следующий блок
; закрываем  файл
CLOSE:
      MOV AH,3EH
      INT  21H
EXIT:
      MOV AH,4CH
      INT  21H
;здесь сообщение о неготовности принтера и
;и ожидание указаний
;если нажимаем Y,  то  выходим  в  DOS
PAUSE:
      MOV AH,9
      LEA  DX,TEXT3
      INT  21H
      MOV AH,0
      INT  16H
      CMP AL,'Y'
      JZ   CLOSE
      CMP AL,'y'
      JZ   CLOSE
      JMP SHORT N1
CODE ENDS
      END BEGIN
```

Рис. 7.8. Вывод на печать текстового файла.

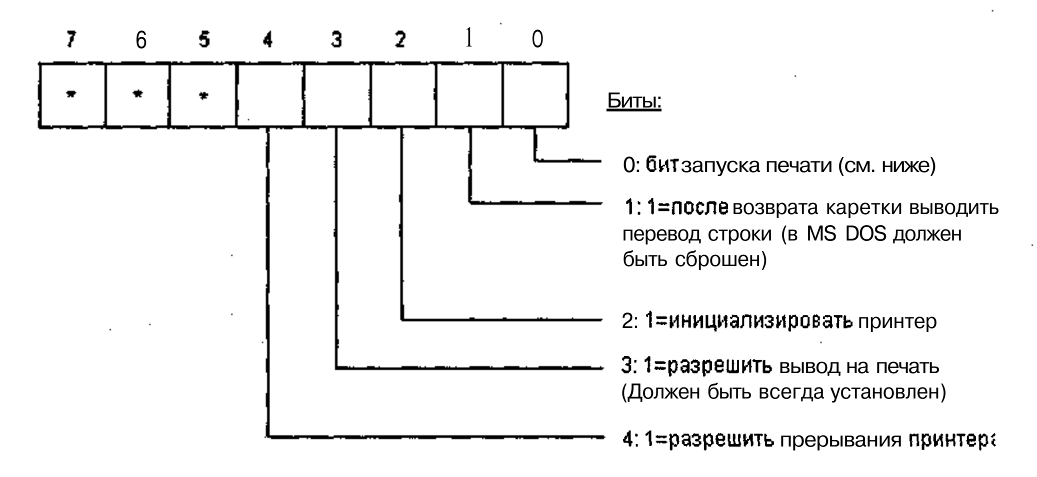
В некоторых ситуациях желательно вообще обойтись без прерывания 17Н и работать только с портами. Рассмотрим в этой связи более подробно порты принтера. Выше было показано, как правильно получить адреса этих портов. Ниже подробно описываются эти порты.

Порт данных - сюда засылается байт, посылаемый на печать. При этом порт может работать и на чтение. В нем хранится последний посланный на принтер байт.

Порт состояния принтера. Работает только на чтение. Первые три бита не используются:



Порт управления принтером. Работает на запись. Последние три бита не используются:



Приведу алгоритм посылки данных на принтер. Я думаю, что читатель без труда сможет реализовать этот алгоритм самостоятельно. Замечу, что речь здесь идет о параллельном интерфейсе в стандарте Центроникс. При работе на отечественной технике (вряд ли Вы ее сейчас встретите) можно встретить и другой стандарт ИРПС, который отличается от предыдущего в значительной степени механизмом посылки строба.

1. Посылаем байт в порт данных.
2. Послать сигнал строба - установить бит 0 в порте управления, а затем сразу сбросить его. Это может быть вначале байт 11, а затем 10.
3. Проверяем, биты 3-5 для обнаружения ошибки. Если ошибка, то переход на процедуру обработки ошибки.
4. Проверяем бит 7. Если принтер готов, то переходим к шагу 1.

В заключение сделаю еще одно, на мой взгляд, важное замечание. Статус принтера можно получить и с помощью функции 2 прерывания 17H. Однако прерывание 17H может перехватывать какой-нибудь драйвер, не совсем корректно обрабатывая информацию. Информацию лучше получать из первых рук. При работе с портом статуса помните, что самыми информативными битами являются 7-й и 4-й. Может статься, что бит 7 всегда будет показывать готовность, тогда вся надежда только на бит 4. Возможна и обратная ситуация, когда о неготовности принтера можно будет судить только по биту 7.

V.

Проблема управления внешними устройствами усложняется тем, что как сами устройства, так и их адаптеры могут несколько отличаться друг от друга. В связи с этим возникает необходимость программного распознавания того, с какими устройствами работает данный компьютер. Мы выходим на совершенно новую тему, которую можно назвать "Ревизией системных ресурсов". Данный вопрос наиболее полно излагается в книге [5] (см. также [9,13]). Мы рассматриваем этот вопрос в главе 23. В данной главе ограничимся лишь перечислением и краткой характеристикой тех средств, с помощью которых можно осуществить такую ревизию.

Функции MS DOS. Среди функций, с помощью которых можно получить различную системную информацию, особо следует выделить функции 1BH, 1CH, 32H, 52H. Из вопросов, которые приходится решать программе с помощью функций DOS, отмечу особенно важные: количество и тип носителей, размер доступной памяти и начало цепочки блоков MCB.

Функции BIOS. Следует особо выделить прерывание 11H, возвращающее слово - список оборудования. Отмечу также прерывания 13H, 10H, через которые можно получить много интересной информации.

Информация на диске. Вместо того чтобы использовать функции DOS, можно посредством прерываний 13H, 25H получить информацию непосредственно с диска (см. главу 14).

Область данных BIOS. Расположение области данных BIOS вы найдете в главе 2, подробную же структуру этой области можно узнать в [13]. Отмечу такие поля, как EGA область, список оборудования (то же, что прерывание 11H), флаги клавиатуры и др.

Порты ввода-вывода. В некоторых случаях без обращения к ним вообще нельзя обойтись. Использование их для диагностики см. в [5]. Описание портов ввода-вывода дано в Приложении 9.

Глава 8. Работа с файлами под управлением MS DOS.

Что имеем - не храним; потерявши - плачем.

Козьма Прутков.

I

Работа с файлами, по правде говоря, единственное, что представлено в системе функций MS DOS достаточно полно. Редко какая программа обходится без обращения к диску. Именно здесь встречается больше всего проблем и ошибок. Я не ставлю своей целью дать всестороннее изложение данного вопроса. Поэтому оставляю в стороне такой вопрос, как работа с файлами методом FCB. Данный метод устарел с того момента, как DOS научилась работать с каталогами. Однако для выполнения условий совместимости он до сих пор представлен в операционной системе. Всех желающих познакомиться с этим подходом отсылаю к замечательной книжке Р. Журдена [5].

Метод работы с файлами в системе MS DOS называется методом описателя или дескриптора. Идея работы с файлами методом описателя в MS DOS заключается в следующем:.

1. Вначале файл должен быть открыт, при этом должно быть указано имя файла - либо полное (полный путь), либо краткое. В последнем случае файл берется из текущего каталога. В конце имени файла должен стоять код 0.
2. После того как файл был удачно открыт, ему присваивается описатель - число от 5 до 256. Дальнейшая работа будет вестись теперь через этот описатель. В PSP программы, по смещению 18H находится таблица описателей. Под нее отводится 20 байт, поэтому программа не может открыть одновременно больше 20 файлов. Байт, содержащий FFH, соответствует свободному описателю. Использование этой таблицы см. в главе 19 и в конце данной главы. Кроме того, в PSP по смещению 32H (слово) хранится размер таблицы описателей, а в четырехбайтовой ячейке со смещением 34H записан полный адрес этой таблицы (по умолчанию 18H и сегментный адрес PSP).
3. В конце работы файл следует закрыть.
4. Помните, что признаком ошибки при выполнении функции DOS является взведенный флаг С (переноса).

Заметим, что описатель файла есть лишь некий индекс, по которому можно найти область памяти, выделяемую для работы с данным файлом. Эта область памяти необходима для того, чтобы буферизовать ввод и вывод в этот файл, что делает работу с ним более быстрой. При записи в файл данные записываются сначала в буфер и, только если он переполнен, записываются на диск. Таким образом, значительно увеличивается скорость записи на диск. При закрытии файла содержимое буфера записи сбрасывается на диск, и далее этот буфер может использоваться для работы с другим файлом. Аналогично работает буферизация при чтении из файла. Читателю, я надеюсь,

теперь понятно, почему данные незакрытого файла могут оказаться потерянными. Должно быть ясно также и то, почему при открытии необходимо указывать режим работы с файлом (чтение, запись, то и другое). Ведь при открытии выделяются буфера отдельно на чтение и запись. В MS DOS имеется функция 68H, с помощью которой можно сбрасывать буфера, выделенные для записи, на диск, не закрывая файла. Эту функцию часто используют для того, чтобы обезопасить себя от возможной потери данных. Аналог ее имеется во всех языках высокого уровня.

Итак, все по порядку.

1. Открыть файл. Рассмотрим фрагмент.

```
LEA DX,PATH ;в сегменте данных путь: C:\ПУТЬ\ИМЯФАЙЛА,0
            ;если краткое имя, то берется текущий каталог
MOVAL,2     ;открыть для чтения  и записи
            ;если 0 для чтения, 1 для записи
MOV AH,3DH  ;функция открытия файла
INT 21H     ;открываем файл
JC  ERRO    ;ошибка, если поднят флаг C, в AX код ошибки
            ;если ошибки нет, то в AX описатель файла
            ;для будущей работы следует его сохранить
```

Имейте в виду, что ошибка обязательно появится, если такого файла не существует (в указанном каталоге, естественно). Но если файла нет, то следует его создать.

Несколько слов об описателях. Количество описателей в системе задается в файле CONFIG.SYS: FILES=N. Причем значение меняется от 5 до 255. Если такой строки нет, то по умолчанию системе дается восемь описателей (8 одновременно открытых файлов, включая и всегда открытые, см. главу 6). Надо, однако, иметь в виду, что количество одновременно открытых файлов (включая предопределенные файлы) не может превышать 20, хотя количество описателей может быть больше. Поскольку 5 первых описателей всегда открыто, то получается, что мы сможем одновременно открыть, на самом деле не более 15 файлов. В конце главы будет показано, как можно решить данную проблему.

2. Создать файл. Фрагмент для создания файла аналогичен фрагменту в пункте 1 (только теперь функция 3CH), но есть и отличие:
   а) файл всегда открывается для чтения и записи;
   б) в CX следует поместить атрибут файла (надеюсь, Вы знаете, что такое атрибут файла), например, 0 или 32 для обычных файлов (напомню, что атрибут 32 означает, что копия с этого файла с помощью программы BACKUP не делалась). Создавая файл, помните, что если файл с таким именем уже существует, то содержимое его будет потеряно. Поэтому, прежде чем создавать файл с помощью данной функции, убедитесь, что в заданном каталоге его нет. Есть, однако, функция 5BH, которая отличается от 3CH только тем, что, если файл уже существует, содержимое его не уничтожается, а взводится флаг ошибки C.

3. Закрыть файл.

```
MOVAH,3EH        ;функция закрытия файла
                 ;в BX должен находиться описатель файла
INT 21H          ;выполняем закрытие
JC  ERROR        ;если флаг взведен, то ошибка
```

При закрытии файла на диск сбрасываются все буфера. Обновляется длина файла, время и дата последней корректировки.

4. Чтение и запись в файл. При работе с файлами на ассемблере помните, что единичной записью является байт. Все записи имеют номера от 0 до L-1, где L длина файла. При открытии файла указатель устанавливается на запись 0. При чтении или записи указатель автоматически передвигается на n байт (где n - число прочитанных или записанных байт). Рассмотрим фрагмент, демонстрирующий запись в файл (чтение из файла — см. главу 6).

```
MOV AH,40H      ;номер функции записи в файл
LEA DX,BUF      ;буфер в сегменте данных
MOVCX,500       ;сколько записывать байт из буфера
JC  ERROR       ;ошибка если флаг взведен и в AX код ошибки
```

При чтении файла (функция 3FH) в AX помещается считанное количество байт. Поэтому следует каждый раз сравнивать AX и CX. Если AX<CX, то обычно это означает, что в процессе чтения произошел переход через конец файла (но не забывайте про флаг C!). При записи в файл ситуация аналогична, но в этом случае неравенство содержимого AX и CX будет означать, что в процессе записи произошла ошибка.

Как я уже говорил, при чтении или записи указатель автоматически передвигается на следующий за считанным блоком байт. Используя функцию 42H, можно переместиться к любому байту файла. Ниже дается полное описание этой функции.

```
AH      42H
BX      описатель файла
CX:DX   на сколько передвинуть: CX*65536+DX
AL      как передвигать
        0 начало файла + CX:DX,
        1 текущий файл + CX:DX,
        2 конец файла + CX:DX
```

Если флаг переноса установлен, то в AX помещен код ошибки, в противном случае AX:DX показывает новую позицию в файле.

С помощью данной функции легко определить длину файла: обнуляем DX и CX и вызываем функцию с AL=2, тогда в AX:DX будет содержаться длина файла.

Вот, в общем, все основные функции для работы с файлами. Надеюсь, что с остальными Вы легко справитесь сами, используя справочник функций DOS в Приложении 7.

Перечислю еще несколько функций, которые без сомнения понадобятся Вам при работе с файлами: 1. Поиск первого вхождения файла, 2. поиск последующего вхождения файла, 3. удалить файл, 4. переименовать файл, 5. изменить или получить атрибут файла, 6. изменить или получить время и дату создания (последней модификации) файла, 7. создать каталог, 8. удалить каталог, 9. сменить каталог. Примеры на использование некоторых из этих функций Вы найдете, как в данной главе, так и далее в книге.

II

Итак, рассмотрим первый пример (Рис. 8.1). Здесь представлена программа вывода на экран последней строки текстового файла. Алгоритм основан на том, что разделителем между строками в текстовом файле является последовательность кодов 13, 10. В этом весь фокус. Мы переходим в конец файла, а затем ищем начало последней строки по коду 10. Код 13 скажет нам о конце строки. Так как у последней строки может не быть приписки 10, 13, то конец в этом случае определим по неравенству содержимого AX и CX.

```
DATA SEGMENT
PATH DB "PRIMER.TXT",0   ;имя файла (текущий каталог)
BUF  DB ?                ;буфер для считывания байта из файла
CX_  DW ?                ;временно храним CX
DX_  DW ?                ;временно храним DX
DATA ENDS
SSEG SEGMENT STACK
     DB 200 DUP(?)
SSEG ENDS
CODE SEGMENT
     ASSUME CS:CODE, DS:DATA, SS:SSEG
BEGIN:
     MOV AX,DATA
     MOV DS,AX               ;DS на сегмент данных
;открываем файл
     MOV AX,3D00H
     LEA DX,PATH
     INT 21H
;если ошибка, то заканчиваем
     JC  EXIT
;на конец файла, определяя длину
     MOV BX,AX
     XOR CX,CX
     XOR DX,DX
     MOV AX,4202H
     INT 21H
;три байта от конца (вдруг в конце тоже стоит 13,10)
     MOV CX,DX
     MOV DX,AX
     MOV AX,4200H
     SUB DX,3
```

```
      SBB  CX,0
      MOV  CX_,CX
      MOV  DX_,DX
      INT  21H
LOO:
;читаем один байт
      MOV  AH,3FH
      LEA  DX,BUF
      MOV  CX,1
      INT  21H
;проверяем, не конец ли предыдущей строки
      CMP  BUF,10
      JZ   OUT_STR
;сдвигаем на один байт к началу
      SUB  DX_,1
      SBB  CX_,0
      MOV  DX,DX_
      MOV  CX,CX_
      MOV  AX,4200H
      INT  21H
      JMP  SHORT LOO
;здесь выводим строку
OUT_STR:
      MOV  AH,3FH
      LEA  DX,BUF
      MOV  CX,1
      INT  21H
      CMP  BUF,13                   ;не конец ли строки?
      JZ   CLOSE
      CMP  AX,CX                    ;не конец ли файла?
      JNZ  CLOSE
; выводим символ
      MOV  DL,BUF
      MOV  AH,2
      INT  21H
      JMP  OUT_STR
CLOSE:                              ;закрыть файл
      MOV  AH,3EH
      INT  21H
EXIT:
      MOV  AH,4CH
      INT  21H
CODE ENDS
      END BEGIN
```

*Рис. 8.1. Вывод на экран **последней** строки текстового файла.*

Следующий пример демонстрирует вывод на экран содержимого текущего подкаталога (имен файлов, кроме скрытых, и подкаталогов). На досуге подумайте, как усовершенствовать программу - вывод по алфавиту, выделение подкаталогов и т.п. Шаблон для поиска может быть Вами изменен, и в соответствии с ним будут выдаваться те или иные имена файлов.

```
DATA SEGMENT
PATH DB "*.*",0               ;шаблон для поиска

DTA DB 50 DUP(?)              ;буфер для размещения текущей
                              ;информации для функций 4EH и 4FH
DATA ENDS
SSEG SEGMENT STACK
     DB 200 DUP(?)
SSEG ENDS
CODE SEGMENT
     ASSUME CS:CODE, DS:DATA, SS:SSEG
BEGIN:
     MOV AX,DATA
     MOV DS,AX
;устанавливаем буфер DTA
     MOV AH,1AH
     LEA DX,DTA
     INT 21H
;вообще говоря можно пользоваться готовым буфером DTA,
;устанавливаемый системой в PSP со смещением 80H
;ищем первое вхождение
CALL FIND_F
JC   EXIT                       ; нет файлов - выходим
DIR:
; ищем последующие вхождения
     CALL FIND_N
     JC   EXIT
;выводим имя
     CALL OUT_NAME
JMP  SHORT DIR
;конец
EXIT:
     MOV AH,4CH
     INT 21H
;процедура поиска первого вхождения
FIND_F PROC
     LEA DX,PATH
     MOV AH,4EH
```

```
;в CX атрибут файла, установим биты так,
;чтобы поиск осуществлялся по всем файлам,
;кроме скрытых и меток тома
     MOV  CX,110101B
     INT  21H
     RET
FIND_F ENDP
;процедура поиска последующих вхождений
FIND_N  PROC
;если после выполнения функции 4EH Вы не меняете DTA
;то следующая строка не обязательна
     LEA  DX,DTA
     MOV  AH,4FH
     INT  21H
     RET
FIND_N ENDP
;процедура вывода имени файла
;это имя расположено в буфере DTA,  по смещению 1EH
OUT_NAME  PROC
     MOV   BX,1EH
     MOV   AH,2
OUT_S:
     MOV   DL,[DTA+BX]
     CMP   DL,0
     JZ    KON
     INT   21H
     INC   BX
     JMP   SHORT OUT_S
KON:
;перевод строки
     MOV   DL,13
     INT   21H
     MOV   DL,10
     INT   21H
     RET
OUT_NAME  ENDP
CODE  ENDS
     END  BEGIN
```

Рис. 8.2. Вывод содержимого подкаталога на экран.

На Рис.8.3 приведена программа, которая осуществляет слияние двух произвольных файлов: к файлу PRIMER2.TXT добавляется файл PRIMER1.TXT. Обращаю Ваше внимание на то, что в различных программах мы берем различные размеры буферов.

Этим подчеркивается, что размер буфера несущественен для представленного алгоритма. Он влияет лишь на быстроту выполнения программы: при большем буфере программа выполняется быстрее.

```
DATA  SEGMENT
PATH1 DB 'PRIMER1.TXT',0 ;имя первого файла
PATH2 DB 'PRIMER2.TXT',0 ;имя второго файла (куда добавлять)
HANDL1 DW ?              ;описатель первого файла
HANDL2 DW ?              ;описатель второго файла
BUFER  DB  1000 DUP(?)   ;буфер
EOF   DB 0               ;если 1, то в процессе чтения
                         ;достигнут конец файла
DATA  ENDS
SSEG SEGMENT STACK
    DB 200 DUP(?)
SSEG ENDS
CODE  SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:SSEG
BEGIN:
    MOV AX,DATA
    MOV DS,AX
;открываем первый файл
    MOV AH,3DH
    MOV AL,0
    LEA  DX,PATH1
    INT 21H
    JC   EXIT
    MOV HANDL1,AX
;открываем второй файл
    MOV AH, 3DH
    MOV AL,1
    LEA DX,PATH2
    INT 21H
    JC   CLOSE1
    MOV HANDL2,AX
;указатель второго файла на конец
    MOV AH,42H
    MOV BX,HANDL2
    XOR CX,CX
    XOR DX,DX
    MOV AL,2
    INT 21H
;готовим регистры
    LEA DX,BUFER
```

```
      MOV CX,1000
;блок копирования
LOO:
;читаем
      MOV  BX,HANDL1
      MOV  AH,3FH
      INT  21H
      CMP  AX,CX
      JZ   NORM
      MOV  CX,AX                       ;<1000 байт
      MOV  EOF,1                       ;достигнут конец файла
NORM:
;пишем
      MOV  BX,HANDL2
      MOV  AH,40H
      INT  21H
      CMP  EOF,0                       ;не достигнут ли конец
      JZ   LOO
;закрываем второй файл
CLOSE2:
      MOV  AH,3EH
      MOV  BX,HANDL2
      INT  21H
;закрываем первый файл
CLOSE1:
      MOV  AH,3EH
      MOV  BX,HANDL1
      INT  21H
;выход в DOS
EXIT:
      MOV  AH,4CH
      INT  21H
CODE  ENDS
      END  BEGIN
```

Рис. 8.3. Слияние двух файлов.

Следующая программа обрабатывает заданный в командной строке файл таким образом, что все прописные латинские буквы преобразуются в заглавные. Блок, анализирующий командную строку, здесь идентичен аналогичному блоку в программе на Рис. 7.6 (Глава 7). В данной программе обратите особое внимание на механизм просмотра файла. Этот механизм в значительной степени основан на том, что размер буфера не превышает 255 байт. Попробуйте изменить программу так, чтобы можно было взять больший размер буфера.

```
CODE SEGMENT
     ASSUME CS:CODE, DS:CODE
     ORG 100H
BEGIN:
     JMP BEG
TEXT1  DB  'Нет параметров.',13,10,'$'
TEXT2  DB  'Файл не найден.',13,10,'$'
PATH   DB  80 DUP(O)        ;путь к файлу
BUF    DB  160 DUP(?)       ;буфер для чтения файла
BEG:
;блок анализа командной строки
     XOR SI,SI
     XOR DI,DI
     MOV DL,1
LOO:
     CMP BYTE PTR [81H+SI],ODH
     JZ  NO_PAR
     MOV AL,[81H+SI]
     CMP AL,' '
     JZ  SPACE
     XOR DL,DL
     MOV  [PATH+DI],AL
     INC DI
     JMP SHORT L001
SPACE:
     OR  DL,DL  ;если DL=0 тогда первый параметр закончился
     JZ  NO_PAR
LOO1:
     INC SI
     JMP SHORT LOO
NO_PAR:
     OR  SI,SI           ;был ли параметр
     JNZ CONT
;сообщение, затем выходим
     MOV DX,OFFSET TEXT1
     MOV AH,9
     INT 21H
     JMP EXIT
;теперь открытие и преобразование файла
CONT:
/открыть файл
     LEA DX,PATH
     MOV AX,3D02H
     INT 21H
```

```
    JNC NORM
    MOV DX,OFFSET TEXT2
    MOV AH,9
    INT 21H
    JMP EXIT
NORM:
    MOV BX,AX
    XOR DI,DI ;в DI будет хранится начало считываемого участка
POVT:
;читать участок файла в буфер
    LEA DX,BUF
    MOV AH,3FH
    MOV CX,160        ;размер буфера
    INT 21H
    MOV AH,AL
    LEA SI,BUF
    CMP AL,0
;просматриваем буфер и преобразуем латинский шрифт
L02:
    JZ  ZER
    CMP BYTE PTR [SI],97
    JB  L01
    CMP BYTE PTR [SI],122
    JA  L01
    SUB BYTE PTR [SI],32
L01:
    INC SI
    DEC AL
    JMP SHORT L02
ZER:
    PUSH AX
;перемещаем указатель файла назад
    MOV AX,4200H
    XOR CX,CX
    MOV DX,DI         ;указатель начала считанного участка
    INT 21H
; пишем буфер на диск
;количество записанных байт может, вообще говоря,
;быть больше 160
    MOV AH,40H
    POP CX
    PUSH CX
    MOV CL,CH
    XOR CH,CH
```

```
      LEA  DX,BUF
      INT  21H
;проверяем, не достигнут ли конец файла
      POP  AX
      MOV  AL,AH
      XOR  AH,AH
      ADD  DI,AX
      CMP  AL,160        ;сравниваем с размером буфера
      JZ   POVT
;закрыть файл
      MOV  AH,3EH
      INT  21H
EXIT:
      RET
CODE  ENDS
      END BEGIN
```

Рис. 8.4. Программа инвертирования прописных латинских символов в файле.

III

Использование описателей файлов таит в себе достаточно интересные возможности. Частично мы уже коснулись этого вопроса, когда выводили информацию на экран при помощи стандартной функции записи в файл (Глава 6, Рис. 6.1). Другие возможности метода описателей связаны с использованием функций 45Н и 46Н. Рассмотрим эти функции.

Дублировать **описатель.**
Вход:
АН - 45Н,
ВХ - описатель (например, вывода на экран).
Выход:
если нет ошибки (флаг не взведен)
АХ - новый описатель иначе код ошибки.

Переназначить описатель.
Вход:
АН - 46Н,
ВХ - уже существующий описатель,
СХ - исходный описатель.
Выход:
АХ - код ошибки, если флаг взведен.

Впервые на практике с этой проблемой я столкнулся лет 10 назад. Мне понадобилось запустить из программы архиватор, причем часть вывода (не информативная)

должна была быть блокирована, а часть должна была выводиться в файл. С помощью указанных функций проблема решалась изящно и просто. Вот алгоритм решения.

1. Дублируем описатель вывода на экран (1).
   ```
   MOV AH,45H
   MOV BX,1
   INT 21H
   ```
   Сохраним описатель.
2. Открываем файл.
3. Переназначаем описатель вывода на экран.
   ```
   MOV AH,46H
   MOV CX,1
   INT 21H   ;при этом в BX должен находиться описатель файла
   ```
4. Закрываем описатель файла. Теперь весь стандартный экранный вывод пойдет в файл.
5. Запускаем архиватор.
6. Переназначаем описатель 1 на старое значение (см. п. 1).
7. Закрываем описатель, полученный в п. 1.

Внимательный читатель заметит, однако, что проблема решена не полностью. Дело в том, что часть информации может выводиться с помощью функций стандартного вывода на экран. В обычных архиваторах так и происходит. В частности так выводится информация о динамике работы архиватора. Эта часть информации может быть блокирована путем перехвата прерывания 10Н, через которое идет весь вывод на экран, разумеется, кроме прямого.

И еще один пример - Рис.8.5.

```
DATA SEGMENT
     PATH DB '1.TXT',0 ;файл для вывода
     BUF DB ?          ;буфер
DATA ENDS
STA SEGMENT STACK
     DB 100 DUP(?)
STA ENDS
CODE SEGMENT
     ASSUME CS:CODE, DS:DATA, SS:STA
BEGIN:
     MOV AX,DATA
     MOV DS,AX
;открываем файл
     MOV AH,3CH
     LEA DX,PATH
     MOV CX,0
     INT 21H
```

```
    JC   KONEC   ;если ошибка - конец
    MOV BX,AX   ;сохранить описатель
;установить буфер
    LEA DX,BUF
READ:
;читаем со стандартного устройства один символ
    PUSH BX
    MOV BX,0
    MOV CX,1
    MOV AH,3FH
    INT 21H
    POP BX
    JC   CLOSE ;если не удалось прочесть, то конец
    MOV SI,CX
    MOV DI,AX
    MOV CX,AX
; пишем в файл
    MOV AH,40H
    LEA DX,PATH
    INT 21H
    JC CLOSE ;если не удалось, то конец
    CMP CX,AX
    JNZ CLOSE
    CMP SI,DI
    JZ   READ
;закрываем файл
CLOSE:
    MOV AH,3EH
    INT 21H
KONEC:
    MOV AH,4CH
    INT 21H
CODE ENDS
    END BEGIN
```

Рис. 8.5. Пример использования описателя стандартного ввода.

Эта простая программа выполняет следующие действия: получает символы со стандартного устройства ввода и отправляет их в файл 1.TXT. Прекратить выполнение программы можно, нажав Ctrl+C. Таким образом, текст можно непосредственно с клавиатуры отправлять в файл. На примере этой программы можно проверить и такое интересное свойство операционной системы MS DOS, как конвейеризация. Пусть оттранслированная программа будет называться INP.EXE. Наберите строку: DIR|INP - в результате в файл 1.TXT будет выведен список файлов текущего каталога.

Как видно из данной главы, работа с файлами всецело основана на функциях DOS. Полный список этих функций приведен в Приложении 7.

IV.

Теперь рассмотрим вопрос о том, как программа может одновременно работать с большим количеством (больше 15) файлов (см. начало Главы). Для этого необходимо:

1. В файл CONFIG.SYS поместить строку FILES=N, где N - нужное количество одновременно открытых файлов. Как мы уже разбирали, этого недостаточно, т.к. таблица файлов находится в PSP и может содержать не более 20 описателей.
2. Скопировать таблицу на новое место и указать ее новый адрес и размер. Следующая программа демонстрирует это.

```
DSEG SEGMENT
     NUM  DB 0
;новая таблица файлов
     DESC DB 50 DUP(0FFH)
;шаблон для открытия файлов
;добавляя в 4-ю и 5-ю позиции различные символы
;мы будем генерировать новые имена файлов
     PATH DB 'FILE',0,0,0
DSEG ENDS
STSEG SEGMENT STACK
     DW 50 DUP(?)
STSEG ENDS
CSEG SEGMENT
     ASSUME CS:CSEG, DS:DSEG, SS:STSEG
BEGIN:
     MOV AX,DSEG
     MOV DS,AX
     MOV NUM,0
;создаем новую таблицу файлов
     CALL OPEN_TAB
;готовим регистры для открытия файлов
     XOR CX,CX
     LEA DX,PATH
;вначале SI+4 указывает на четвертую позицию в имени файла
     MOV SI,DX
     MOV BL,65
;попытка открыть одновременно 40 файлов
LOO:
     MOV AH,3CH
     MOV [SI+4],BL
     INT 21H
```

```
    JC  _END   ;если ошибка создания, то конец
    CMP BL,'Z'
    JNZ N0
    INC SI
; теперь SI+5 указывает на 5-ю позицию в имени файла
    MOV BL,65
NO:
    INC BL
    INC NUM
    CMP NUM,40
    JNZ LOO    ;если не сорок, то продолжим
_END:
;вывести, сколько файлов было реально открыто
    MOV  AL,NUM
    CALL DISP_BYTE
    MOV AX,4C00H
    INT 21H
;область процедур
;процедура вывода байта в AL
DISP_BYTE PROC NEAR
    XOR AH,AH
    MOV BL,100
    DIV BL
    MOV  SI,AX
    MOV  AL,AH
    XOR  AH,AH
    MOV  BL,10
    DIV  BL
    MOV  BX,AX
    MOV  DX, SI
    MOV  AH, 2
    ADD  DL,48
    INT  21H
    MOV  DL,BL
    ADD  DL,48
    INT  21H
    MOV  DL,BH
    ADD  DL,48
    INT  21H
    RETN
DISP_BYTE ENDP
;процедура открытия новой таблицы файлов
OPEN_TAB PROC NEAR
```

```
;вначале копируем таблицу файлов на новое место
     MOV  CX,20
     LEA  DI,DESC
     MOV  SI,18H
LOO1:
     MOV  AL,ES:[SI]
     MOV  DS:[DI],AL
     INC  SI
     INC  DI
     LOOP LOO1
;здесь указываем новый размер и адрес таблицы
;размер таблицы описателей
     MOV  WORD PTR ES:[32H],50
;положение таблицы
     MOV  ES:[36H],DS
     MOV  WORD PTR ES:[34H],OFFSET DS:DESC
     RETN
OPEN_TAB ENDP
CSEG ENDS
     END BEGIN
```

Рис. 8.6. Пример использования своей таблицы файлов.

Данная программа должна одновременно открывать 40 файлов при условии, что в файле CONFIG.SYS стоит строка FILES=N, где N>45. При окончании работы программа выводит количество одновременно открытых файлов. При успешном завершении это число равно 40. Процедура создания новой таблицы файлов называется **OPEN_TAB**. Если отключить эту процедуру, то количество одновременно открытых файлов будет равно 15 (или даже меньше). Поэкспериментируйте с этим. Кстати, приблизительно так работают многие системы управления базами данных для операционной системы MS DOS, которые позволяют одновременно работать с большим количеством открытых файлов. Новую таблицу файлов мы поместили в сегмент данных программы, но с тем же успехом могли бы выделить для этой таблицы область памяти динамически (см. управление памятью в Главе 11).

Сходным образом проблема работы с большим количеством одновременно открытых файлов решается посредством функции 67H DOS (см. Приложение 7). Данная функция автоматически выделяет место для новой таблицы и возвращает указатель на эту область (сегментный адрес в ES).

Глава 9. Прерывания.

Граждане, этот больной пойдет вне очереди,

М.А. Булгаков
Мастер и Маргарита.

I.

Прерывание по своему смыслу есть временное прекращение какого-то процесса. Команды INT и CALL реализуют программные прерывания. Они выполняются, когда приходит время выполнить соответствующую команду. После их выполнения программа продолжает работать с команды, стоящей за командой вызова прерывания. Существуют и аппаратные прерывания, которые происходят, когда наступает некоторое событие, внешнее по отношению к программе. Это может быть сигнал по прошествию определенного промежутка времени, нажатие клавиши, переход принтера в состояние готовности, наступление некоторого события в микропроцессоре (деление на нуль, переполнение и т.п.) и т.д. Соответственно, аппаратные прерывания, происходящие от внешних устройств, будем называть внешними, а аппаратные прерывания, происходящие от события в микропроцессоре - внутренними. Есть еще немаскируемое прерывание - NMI. Это прерывание невозможно запретить командой CLI. Мы не будем больше рассматривать NMI, за исключением материала главы 26.

Выполнять команды CALL и INT мы научились. А как использовать прерывания в своих целях? Здесь многому еще придется научиться.

Как было указано в главе 1, в начале памяти (младшие адреса) располагаются векторы прерываний. Некоторые векторы устанавливаются до загрузки операционной системы, некоторые устанавливает сама MS DOS. Например, вектор 8H (т.е. в памяти он расположен по адресу 0:8H*4, см. главу 2) является вектором прерывания таймера. При нормальной установке каждую секунду происходит 18.2 обращений к процедуре обработки, на которую указывает это вектор. Нет принципиального различия между векторами программных прерываний (векторы 21H, 16H, 10H и др.) и векторами прерываний аппаратных (это векторы 1H, 9H, 8H и др.). И те, и другие могут стать объектом перехвата, т.е. переустановки на другую процедуру. Зачем это может понадобиться? А затем, что в некоторых ситуациях Вы можете быть недовольны тем, как обрабатывает прерывания системная процедура, и захотите, чтобы обработку вела собственная процедура.

Ничто не мешает установить вектор прерывания на Вашу процедуру. Возникает вопрос: как поступить стой процедурой, на которую он был направлен до перехвата? Здесь может быть три варианта решения. Все остальные будут являться комбинациями этих трех. На Рис. 9.1 эти ситуации показаны схематично.

Из схем, приведенных на Рис. 9.1, схема (С) реже всего встречается в чистом виде. Как правило, на практике встречается некоторая смесь - АС и ВС. Теперь о том, почему желательно, а во многих случаях необходимо выполнять процедуру, на которую указывал старый вектор прерывания. Представьте, что Вы перехватили прерывание номер 21H. Если теперь не выполнять старую процедуру, то функции MS DOS будут заблоки-

рованы. В большинстве случаев такое блокирование нежелательно. И в конце концов надо иметь уважение к тем, кто перехватил данное прерывание раньше и желает получить свое. При таком подходе образуется целая цепочка процедур, которые будут выполняться друг за другом, пока не будет выполнена последняя, после чего произойдет возврат туда, откуда это прерывание было выполнено. Есть, однако, прерывания, которые иначе, чем по схеме (С), нет смысла перехватывать. О них речь пойдет впереди.

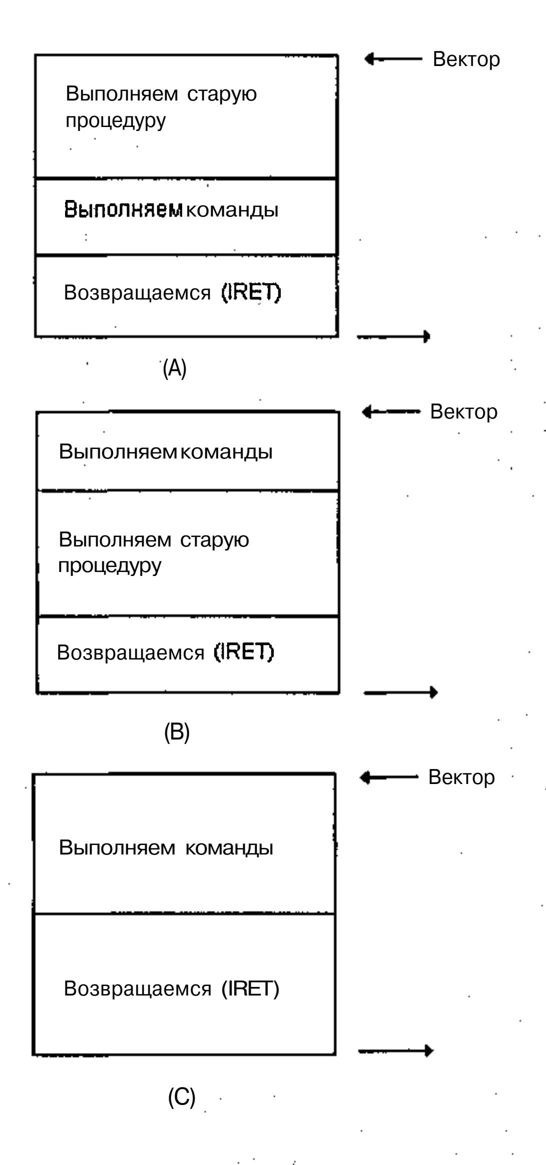


Рис. 9.1. Возможные схемы перехвата прерываний.

Вызвать старую процедуру прерывания можно двумя способами [22]. Пусть старый вектор прерывания хранится в двух смежных словах: O_INT и S_INT:

O_INT - смещение, находится в младшем слове,

S_INT - сегмент, старшее слово; предполагается, *что* слова расположены в сегменте данных.

Первый способ:

```
JMP DWORD PTR DS:[O_INT]
```

передаст управление старой процедуре прерывания, причем возврата в Вашу процедуру не произойдет.

Второй способ:

```
PUSHF
CALL DWORD PTR DS:[O_INT]
```

передаст управление старой процедуре прерывания, после выполнения процедуры (или целой цепочки процедур) произойдет возврат в Вашу программу [23]. Дополнительная PUSHF необходима для того, чтобы правильно был осуществлен возврат по IRET. Второй способ часто бывает необходим, но пользоваться им следует с большой осторожностью. Дело в том, что процедуры прерывания, к которым Вы обратились посредством команды CALL, могут возвращать в регистрах некоторую информацию, например, в регистре флагов, в регистре AX. Вы не должны портить эту информацию (см. главу 12).

Для вызова прерывания можно также использовать свободные векторы. Для этого необходимо направить неиспользуемый вектор (см. ниже) на нужную процедуру и затем вызвать его командой INT. Например, Вы работаете с вектором 16H. Он направлен на Вашу процедуру. Старое значение вектора 16H присвойте, например, вектору FEH. В конце Вашей процедуры обработки поставьте команду INT FEH. Такой вызов будет аналогичен вызову через CALL (см. выше).

Перенаправить вектор можно также двумя способами:

1) используя функции DOS 25H и 35H;
2) непосредственно обратившись к таблице векторов и изменив содержимое соответствующих ячеек.

Рассмотрим первый способ, иллюстрируемый следующим фрагментом.

---

[22] Во всяком случае, использовать для этих целей скажем RETF было бы слишком экзотично.

[23] Вообще говоря, возможны ситуации (довольно редкие), когда возврата из старой процедуры не происходит.

```
...
MOV AH,35H          ;получить вектор прерывания
MOV AL,5           ;вектор 5 (печать экрана) :
INT 21H             ;после выполнения содержимое
                    ;вектора в ES:BX
MOV OLD_S,ES        ;сохранить старый вектор
MOV OLD_O,BX
MOV AH,25H          ;установить вектор на свою процедуру
MOV DX,OFFSET P_5  ;смещение
MOV AX,SEG P_5      /сегмент
MOV DS,AX
INT 21H
...
```

Следующий фрагмент делает то же самое, но при этом не использует функции DOS.

```
...
CLI                  /запретить прерывания
MOV AX,0
MOV ES,AX
MOV DX,ES:[5H*4]    /смещение в DX
MOV BX,ES:[5H*4+2]  /сегмент в BX
MOV OLD_S,BX         ;сохраняем старый
MOV OLD_O,DX         ;вектор
MOV DX,OFFSET P_5    ;
MOV AX,SEG P_5       ;
MOV ES:[5H*4],DX     /изменяем вектор
MOV ES:[5H*4+2],AX
STI                  ;разрешить прерывание
...
```

Обратите внимание, что во втором фрагменте перед тем, как изменять вектор, мы запрещаем прерывания. Это делается для того, чтобы обезопасить себя от прерывания именно по этому вектору, когда он еще не до конца изменен. Функции MS DOS делают это сами.

П.

При работе с прерываниями нужно быть аккуратным. Перед выходом из программы следует присвоить векторам их прежние значения. При выходе в MS DOS область памяти, занимаемая программой, освобождается. Если Вы не присвоите векторам их старые значения, то они будут направлены на освобожденную область памяти, что в конечном итоге приведет к "зависанию" всей системы. Исключение составляют только векторы 23H и 24H (а также 22H) - операционная система сама направит их, куда надо, по выходу из программы.

В главе 7 уже говорилось о том, какую функцию реализует вектор 23H. Если Вы хотите избавиться от Ctrl Break (Ctrl C), направьте его на процедуру, где будет стоять лишь одна команда - IRET. Можно вставить туда и другие команды. Лично я, однако, не использую этот способ (см. главу 7).

Поговорим теперь о векторе 24H [24]. Процедура, на которую указывает данный вектор, выполняется при критических ошибках (отсутствие дискеты, неготовность принтера и т.п.). Вы можете направить его на свою процедуру. Использование вектора 24H открывает довольно широкие возможности обработки критических ситуаций. Зесь будет рассказано только о наиболее частом способе обработки. Перед тем как возвратиться из процедуры, пошлите в регистр AL код обработки: 0 - игнорировать ошибку (опасно); 1 - повторить операцию (можно сделать несколько повторений); 2 - выйти через вектор 23H; 3 - вернуться с индикацией ошибки.

Наиболее приемлемым является третий вариант, который, кстати, чаще всего и используют. Пока не вернетесь из прерывания, не вздумайте обращаться к каким-либо функциям MS DOS. Помните и еще об одном нюансе: при возвращении с AL=3 следует выполнить какую-нибудь функцию MS DOS с номером выше 0CH. Это вернет операционную систему в нормальное состояние. Не пугайтесь, что при этом будет взведен флаг ошибки. Следующие функции MS DOS будут уже выполняться нормально. В последнем разделе главы мы снова возвращаемся к прерыванию 24H.

Рассмотрим конкретный пример перехвата прерываний. На Рис.9.1 представлена программа, выход из которой можно осуществить только по нажатию клавиши ESC. Ни на Ctrl Break, ни на Ctrl C программа не реагирует, несмотря на то, что ввод и вывод осуществляется посредством функций MS DOS (выводится на экран 0 и проверяется буфер клавиатуры). Подход является общим, хотя и непростым. Блокирование нажатия Ctrl C можно проводить и по-другому, сбрасывая флаг клавиши и, выполняя стандартную процедуру обработки прерывания клавиатуры. Я же хотел обратить Ваше внимание на возможность ситуации, когда Вашей программе придется брать на себя всю обработку прерывания, не вызывая стандартную процедуру. Строки, отмеченные звездочками, можно было бы с тем же успехом заменить всего одной строкой (!):

```
AND    BYTE PTR ES:[417H],11111011B .
```

В этом случае нажатие Ctrl C было бы равносильно просто нажатию C.

Отключены в этой программе также клавиши PrtSc и Pause. Первая клавиша отключается путем направления вектора печати экрана (номер 5) на процедуру, содержащую только команду IRET. Отключение клавиши Pause осуществляется путем сбрасывания соответствующего бита по адресу 0:418H (см. главу 7, флаги клавиатуры).

Обратите также внимание на использование в программе портов с адресами 60H и 61H. Это порты микросхемы управления периферией. Мы встречались с портом 61H, когда говорили о генерации звука. Это порт и для чтения, и для записи. Засылка в порт

[24] Я не ставил своей задачей рассказать обо всех векторах. О них с исчерпывающей полнотой Вы прочтете в справочнике программиста. Я хочу лишь изложить основные идея, а как Вы их будете применять, Ваше дело.

байта с установленным 7-м битом дает подтверждение, что скан-код нажатой клавиши принят. Порт 60H используется только для чтения и после нажатия клавиши содержит скан-код этой клавиши.

```
DATA SEGMENT
;в сегменте данных хранятся старые векторы
;вектор прерывания клавиатуры
OLD_09_0 DW ?
OLD_09_S DW ?
;вектор прерывания по Ctrl Break
OLD_1B_O DW ?
OLD_1B_S DW ?
;вектор прерывания печати экрана
OLD_05_0 DW ?
OLD_05_S DW ?
DATA ENDS
SSEG SEGMENT STACK
     DB 200 DUP(?)
SSEG ENDS
CODE SEGMENT
     ASSUME CS:CODE, DS:DATA, SS:SSEG
BEGIN:
     MOV AX,DATA
     MOV DS,AX
;определяем значение векторов 09H, 1BH, 05 сохраняем их
     MOV AX,351BH
     INT 21H
     MOV OLD_1B_S,ES
     MOV OLD_1B_O,BX
     MOV AX,3509H
     INT 21H
     MOV OLD_09_S,ES
     MOV OLD_09_O,BX
     MOV AX,3505H
     INT 21H
     MOV OLD_05_S,ES
     MOV OLD_05_O,BX
;устанавливаем новое значение векторов
     PUSH DS
     PUSH CS
     POP  DS
     MOV  DX,OFFSET CS:INT09
     MOV  AX,2509H
     INT. 21H
     MOV. DX,OFFSET CS:,INT1B
     MOV  AX,251BH
```

```
     INT   21H
     MOV   DX,OFFSET CS:INT05
     MOV   AX,2505H
     INT   21H
     POP   DS
;цикл, из которого можно выйти только нажатием клавиши ESC
LOO1:
     MOV DL,48
     MOV AH, 2
     INT 21H
     MOV AH,06H
     MOV DL,0FFH
     INT 21H
     JZ   LO01
     CMP AL,27
     JNZ  LO01
;восстанавливаем старое значение векторов
     PUSH DS
     POP  ES
     MOV   DX,ES:OLD_09_0
     MOV   DS,ES:OLD_09_S
     MOV   AX,2509H
     INT   21H
     MOV   DX,ES:OLD_1B_O
     MOV   DS,ES:OLD_1B_S
     MOV   AX,251BH
     INT   21H
     MOV   DX,ES:OLD_05_O
     MOV   DS,ES:OLD_05_S
     MOV   AX,2505H
     INT   21H
;выходим в DOS
     MOV AH,4CH
     INT 21H
;процедура прерывания клавиатуры
INT09 PROC FAR
     PUSH AX
     PUSH ES
     PUSH DS
;восстановить значение DS нужно,  потому что вызов
;прерывания клавиатуры может произойти в момент,
/когда DS не указывает на сегмент данных
; (например, во время выполнения функции DOS)
     MOV   AX,DATA
     MOV   DS,AX
     MOV AX,0
```

```
     MOV  ES,AX
     IN AL,60H
     CMP AL,46                          ;скан-код клавиши C
     JNZ LOO
     TEST BYTE PTR ES:[417H],4   ;проверяем, не нажата ли Ctrl
     JZ   LOO
;следующие команды необходимы, чтобы дать контроллеру
;клавиатуры сигнал, что символ принят
     IN    AL,61H     ;*
     MOV   AH,AL      ;*
     OR    AL,80H     ;*
     OUT   61H,AL     ;*
     MOV   AL,AH      ;*
     OUT   61H,AL     ;*
;две следующие команды дают сигнал контроллеру прерываний,
;что процедура прерывания завершена
     MOV   AL,20H     ;*
     OUT   20H,AL     ;*
     POP   DS         ;*
     POP   ES         ;*
     POP   AX         ;*
     IRET             ;*
LOO:
     PUSHF
     CALL DWORD PTR DS:[OLD_09_O]
     AND BYTE PTR ES: [418H],11110111B  ;блокирование клавиши
     ;Pause
     POP DS
     POP ES
     POP AX
     IRET
INT09 ENDP
/процедура прерывания по Ctrl Break
INT1B PROC FAR
     IRET
INT1B ENDP
;процедура прерывания PrtScr
INT05 PROC
     IRET
INT05 ENDP
CODE ENDS
     END BEGIN
```

Рис. 9.1. Пример программы с перехватом прерываний.

IV.

Внешние аппаратные прерывания обслуживает специальное устройство, называемое контроллером прерываний (Intel 8259) (см. Приложение 9). Речь идет о прерываниях, связанных с событиями на внешних устройствах, т.е. внешними прерываниями, но не с событиями, происходящими внутри микропроцессора (деление на нуль и т.п.). Контроллер распознает прерывание и посылает сигнал микропроцессору. Если прерывания разрешены, то микропроцессор на следующем шаге получает байт-номер прерывания и, закончив выполнение текущей команды, выполняет соответствующую команду INT. В задачу контроллера прерываний входит также обслуживание прерываний по приоритетам. Может статься, что, пока обслуживается одно прерывание, произойдет другое. Контроллер определяет приоритет, так что прервать уже выполняющееся прерывание может лишь прерывание с более высоким приоритетом. Если два прерывания произойдут одновременно, то первым будет выполняться прерывание с более высоким приоритетом. Надо, однако, учесть, что, когда выполняется аппаратное прерывание, запрещаются все прерывания на уровне микропроцессора - выполняется команда CLI. Если у Вас нет возражений на то, чтобы прерывания все-таки происходили, поставьте в начале процедуры команду STI. В этом случае будут происходить лишь прерывания более высокого приоритета (этим будет управлять контроллер прерываний). В конце процедуры следует указать контроллеру, что прерывание закончено (см. программу на Рис. 9.1).

Наиболее высоким приоритетом обладает прерывание таймера - вектор 08H. Следует избегать ситуаций, когда это прерывание оказывается надолго запрещенным. Это может повлиять на ход системных часов. Ниже все прерывания расположены в порядке их приоритетов.

Прерывание	Вектор	Источник
IRQ0	8	Таймер
IRQ1	9	Клавиатура
IRQ2	0AH	Канал ввода-вывода
*IRQ8	70H	Часы реального времени (AT)
*IRQ9	71H	Программно переводится в 2
*IRQ10	72H	Резерв
*IRQ11	73H	Резерв
*IRQ12	74H	Резерв
*IRQ13	75H	Мат. Сопроцессор (AT)
*IRQ14	76H	Контроллер жесткого диска (AT)
*IRQ15	77H	Резерв

IRQ3	0BH	COM2 (для XT было COM1)
IRQ4	0CH	COM1 (для XT было COM2)
IRQ5	0DH	LPT2 (для XT был жесткий диск)
IRQ6	0EH	Гибкий диск
IRQ7	0FH	LPT1

Рис. 9.2. Приоритеты прерываний.
Первая колонка показывает номера приоритетов прерываний, вторая - номера векторов, их обслуживающих. Звездочкой отмечены прерывания, которые обслуживаются вторым контроллером.

Из рисунка видно, что у AT имеется больше прерываний, чем у XT. Объясняется это тем, что у AT прерывания обслуживают две микросхемы 8259. Основной контроллер называется ведущим, дополнительный контроллер - ведомым. Вектор прерывания получается путем сложения базового вектора и номера прерывания. У ведущего контроллера базовый вектор равен 8, у ведомого - 70H. Таким образом, номер вектора прерывания от клавиатуры равен 1+8, т.е. 9. Базовое значение вектора прерываний может быть изменено (см. главу 20). Программирование ведущего контроллера осуществляется через порты 20H-21H, ведомого — через A0H-A1H.

На уровне контроллера возможно маскирование отдельных прерываний. Для этого необходимо послать цепочку соответствующих битов в порт 21H - регистр маски (порт для второй микросхемы AT имеет адрес 1AH). Чтобы, например, замаскировать прерывание клавиатуры, достаточно выполнить следующие команды:

```
MOV AL,00000010B
OUT 21H,AL
```

В конце программы следует порт очистить. Если Ваша процедура перехватывает аппаратное (sic!) прерывание (см. Рис. 9.2) по схеме (С), т.е. полностью заменяется стандартный обработчик, то в конце процедуры следует поставить следующие магические строки:

```
MOV AL,20H
OUT 20H,AL (OUT 0A0H,AL - для второго контроллера).
```

Этим Вы очищаете регистр обслуживания прерывания и разрешаете обработку прерываний с более низким приоритетом (см. программу на Рис. 9.1). При перехвате не аппаратных прерываний (не от внешних устройств) в таких действиях нет необходимости. Так, прерывание IRQ8 (от таймера) требует сигнала окончания. Прерывание же с вектором 1CH вызывается уже после отработки первого прерывания и сигнала окончания не требует.

На Рис. 9.3 приведена схема работы контроллера прерываний.

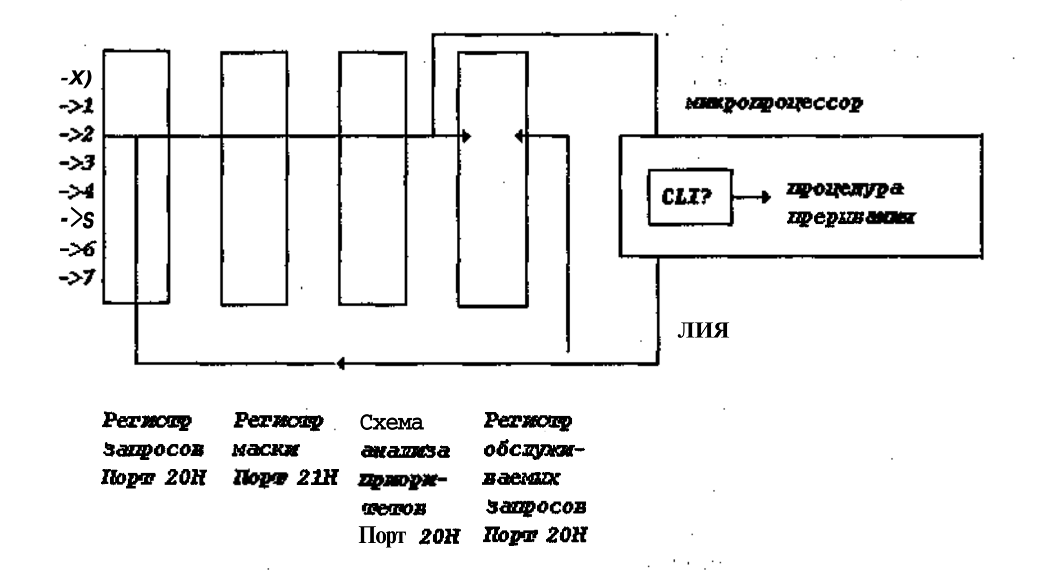


Рис. 9.3. Работа контроллера прерываний.

Рассмотрим работу контроллера прерываний по схеме Рис.9.3. Сигнал запроса прерываний от некоторого устройства (у нас оно имеет номер *2)* поступает на вход регистра запросов и устанавливает в единицу значение соответствующего бита. Пока этот бит не сброшен, прерывание данного номера игнорируется. Далее сигнал поступает в регистр маски. Пройти через регистр маски сигнал может лишь при условии, что соответствующий бит равен нулю. Заметим, что уже пояснялось, как замаскировать отдельные прерывания. Из сказанного следует, что маскирование прерывания сохраняет последний запрос. После регистра маски сигнал поступает на схему анализа приоритетов. Данная схема начинает играть важную роль лишь тогда, когда два или более прерываний накладываются друг на друга. После схемы анализа приоритетов сигнал поступает в микропроцессор и одновременно в регистр обслуживания запросов, где дается разрешение на установку соответствующего бита.

Микропроцессор воспринимает поступивший сигнал лишь в том случае, если разрешены прерывания (не было команды CLI). Если же была команда CLI, то запрос блокируется, однако, как ясно из схемы, не пропадает. Соответственно после разрешения прерываний повторяется. Если прерывания разрешены, то:

1. Посылается сигнал в регистр обслуживания запросов и устанавливается соответствующий бит. Наличие бита будет говорить контроллеру, что в данный момент обслуживается такое-то прерывание.
2. Посылается сигнал в регистр запросов и сбрасывается соответствующий бит. После этого контроллер уже может воспринимать сигнал того же прерывания.
3. Запрещаются Прерывания на уровне микропроцессора - неявно выполняется команда CLI.
4. Выполняется процедура прерывания.

Из изложенной схемы становится ясно, как реагирует контроллер на одновременный приход нескольких прерываний. Если одно прерывание уже работает, то начинает играть схема приоритетов прерываний. Пропускаются лишь прерывания более высокого приоритета, которые, однако, задерживаются на уровне контроллера, т.к. выполнена команда CLI. Если Вы хотите, чтобы Ваша процедура прерывания не блокировала выполнение более высоких прерываний, выполните в начале процедуры команду STI. Более же низкие по приоритету прерывания разрешаются только после посылки в порт 20H числа 20H (см. выше).

Как уже было сказано, базовый вектор для ведущего контроллера равен 8H, а для контроллера ведомого - 70H. Значение этих векторов можно изменить. Тем самым мы можем менять значения векторов прерываний. В главе 26 Вы увидите, что действительно иногда это необходимо сделать. Для того чтобы сменить значение базовых векторов, необходимо провести инициализацию контроллера. Не вдаваясь в подробности, рассмотрим алгоритм такой инициализации.

1. **Послать в порт 20H (A0H для второго контроллера) число 11H (для AT).**
2. Послать значение базового вектора в порт 21H (A1H для второго контроллера).
3. Послать слово, определяющее, к какому входу ведущего контроллера подсоединен контроллер ведомый. Обычно это вход 2. Биту 2 соответствует число 4, которое посылается в порт21H (и в A1H).
4. Послать в порт 21H (в A1 H) число 1.

Этот алгоритм будет использован в дальнейшем в главе 26.

Подведем некоторый итог:

1. Сигнал прерывания от внешнего устройства попадает вначале в контроллер прерываний, который, как хороший секретарь, пропускает сигналы к микропроцессору по очереди и по приоритетам.
2. Номер пришедшего прерывания посредством базового вектора определяет вектор прерывания, а следовательно, и вызываемую процедуру. Базовый вектор может быть изменен.
3. Существует три уровня запрета прерываний:
   а) на уровне микропроцессора (CLI),
   б) на уровне контроллера прерываний (см. выше),
   в) на уровне конкретного устройства - запретить вырабатывать сигналы прерываний.

V.

Мир прерываний весьма разнообразен, как и использование их в программах. Здесь перечисляются некоторые (не все) прерывания вместе с тем, как их обычно используют программисты, перехватывая в своих программах. Дальнейшие примеры использования прерываний Вы найдете в следующих главах и в последнем разделе настоящей главы.

Номер	Назначение
0	Происходит при возникновении ошибки «переполнение при делении»[25]. Чтобы это не происходило, можно перехватить его и направить на IRET. Более правильно будет включить его в алгоритм деления (см. [3]), а также программу ниже (Рис. 9.5).
1	После установки бита трассировки микропроцессор выполняет его после каждой команды. Перехватывается различными отладчиками. Если Вы захотите выяснить первоначальное значение какого-либо вектора, без перехвата прерывания 1 не обойтись.
5	Вектор процедуры печати экрана. Вызывается из прерывания клавиатуры. Можете перехватить его, чтобы установить свою процедуру печати экрана. В своих программах я люблю просто отключать его, направляя на IRET.
8	В нормальном состоянии вызывается 18.2 раз в секунду. Одна из задач этого прерывания - поддерживать правильный ход системных часов. Часто перехватывается различными программами для выполнения фоновых задач. Вызывает пользовательское прерывание 1CH. Выключает мотор дисководов, если в течение двух секунд к дисководу не было обращения.
9	Прерывание клавиатуры. Резидентными программами используется для взведения флага вызова. Можно перехватить для распознавания нажатия таких клавиш, как Shift, Alt, Ctrl, CapsLock и т.д.
0CH	Прерывание, вызываемое при совершении события в порте COM1. Перехватывается в различных коммуникационных программах.
OBH	Прерывание, вызываемое при совершении события в порте COM2. Аналогично предыдущему.
0FH	Прерывание принтера. Вызывается, когда принтер готов к выполнению очередной операции. Используется редко, так как не все адаптеры его поддерживают.
ЮН	Данный вектор обычно перехватывают драйверы экрана для отслеживания различных экранных операций. Я иногда использую перехват этого прерывания для блокирования Вывода на экран.
11H	Определение набора подключенного оборудования (ненадежно).
13H	Дисковые операции. Перехватываются некоторыми резидентными программами для контроля над операциями ввода-вывода.
14H	Функция обслуживания порта асинхронной связи.

[25] Для микропроцессоров 8086/8088 программа не прерывалась, а после сообщения продолжала выполняться следующая инструкция.

15H	Расширенный AT-сервис (см. главу 5)
16H	Процедуры обслуживания клавиатуры. Иногда вместо вектора 9 удобнее и безопаснее работать с этим.
17H	Процедура вывода на печатающее устройство. Перехватывается различными резидентными драйверами печати. Особенно часто это приходится делать в случае несовпадения кодировки принтера и компьютера.
18H	В некоторых компьютерах при вызове этого прерывания запускается кассетный Бейсик.
19H	Вызывает процедуру загрузки ОС.
1AH	Функции таймера и часов реального времени.
1BH	Прерывание вызывается при нажатии Control_Break.
1CH	Пользовательское прерывание таймера. Вызывается из 8-го. Не требует работы с контроллером прерываний. Используется для фоновых задач: фоновая музыка, часы, активизация резидентных программ и т.д.
21H	Функции DOS. Перехватывается резидентными программами для реализации их безопасного вызова. Перехватывается различными антивирусными мониторами.
23H	Выход по Ctrl Break. О перехвате его мы говорили в главе 7.
24H	Критическая ошибка. Перехватывается для собственной обработки критических ошибок.
25H-26H	Процедуры DOS прямой записи-чтения диска. Перехватываются для контроля над обращением к диску.
27H	Процедура "Остаться резидентным". Перехватывается различными антивирусными мониторами.
28H	Недокументированное прерывание. Используется для безопасного вызова резидентных программ.
2FH	Мультиплексное прерывание. Специально используется резидентными программами для "общения" друг с другом. Первоначально предполагалось для использования утилитой MS DOS PRINT.EXE.

Наиболее эффективной формой использования аппаратных прерываний является организация фонового ввода-вывода. Программа должна знать, когда можно передавать или принимать данные. Можно пойти по пути непрерывного опроса. Однако если учесть, что микропроцессор много быстрее внешних устройств, то получается значительная потеря времени. Если же программа перехватит соответствующее аппаратное прерывание, то микропроцессор будет обращаться к внешнему устройству только при

возникновении соответствующего события - готовности устройства к передаче или приему данных.

Наиболее часто такой подход используется при работе с последовательным портом. Это и работа с мышью, и модемная связь, и связь между компьютерами непосредственно через порт (нуль-модем) и т.д. Посылка данных на принтер по прерыванию также возможна. К сожалению, не все адаптеры параллельного порта способны работать по прерыванию, поэтому такой подход используется довольно редко. Для организации фоновой печати применяется другой метод. Опрос параллельного порта осуществляет процедура, вызываемая прерыванием по времени (вектора 08H или 1CH). Кстати, именно таким образом работает известная DOS'овская утилита PRINT.EXE.

Ниже схематично показана процедура передачи данных на печатающее устройство по прерыванию.

```
INT_PRINT   PROC
{сохранить все используемые регистры}
MOV  DS,SEG BUF        ;DS на сегмент, где находится буфер
CMP  LEN_BUF,0         ;не пуст ли буфер
     JZ    QUIT
     CALL  OUTPUT      ;вывод символа, если ошибка, то флаг C
                       ;взведен
     JC    QUIT
     CALL  MOV_POINT   ;переместить указатель в буфере на
                       ;следующий символ
QUIT:
  {восстановить регистры}
     IRET
INT_PRINT   ENDP
```

Данную схему следует дополнить некоторыми пояснениями. В частности, это касается устройства буфера. Часто для этой цели используют кольцевой буфер, наподобие клавиатурного (см. главу 7). Возможен и другой вариант. Его можно назвать буфером с фиксированным началом. Состояние буфера определяет его длина. При этом символы поступают туда по принципу стека. Длина буфера показывает смещение последнего положенного туда символа. Символы берутся с начала (смещение 0). После каждого считывания все символы сдвигаются к началу, и длина буфера уменьшается на единицу. Добавлю также, что, для того чтобы готовность принтера генерировала прерывание, следует установить бит 4 в регистре управления принтера (см. главу 7).

VI.

В этом разделе рассматривается пример взаимодействия по прерыванию с адаптером асинхронной связи (последовательным портом). Программу, которая приводится ниже, легко проверить, если к Вашему компьютеру подключена мышь. Наличие или отсутствие драйвера мыши при этом не имеет никакого значения. Обращаю внимание на то, что мы вначале разрешили прерывание от COM-порта посредством команды кон-

троллеру прерываний, а затем посредством команды адаптеру асинхронной связи. При движении мыши и нажатии кнопок на экран будет выдаваться содержимое регистра данных адаптера. В принципе по этим данным можно было бы отслеживать положение курсора. Программа является типичным примером фоновой программы. Процедура проверки порта адаптера и вывода его содержимого выполняется лишь при получении прерывания от этого адаптера.

```
DATA  SEGMENT
;старый вектор прерываний от адаптера асинхронной связи
OLD_VEC_OFF DW ?
OLD_VEC_SEG DW ?
STR DB 0,0,0,13,10,'$' ;строка для вывода содержимого порта
DATA  ENDS
ST1  SEGMENT  STACK
     DB 200 DUP(?)
ST1 ENDS
CODE  SEGMENT
     ASSUME CS:CODE, DS:DATA, SS:ST1
BEG:
;начальный блок
     MOV AX,DATA
     MOV DS,AX
     MOV AX,ST1
     MOV SS,AX
;установка вектора прерываний
     MOV AX,0
     MOV ES,AX
     MOV BX,WORD PTR ES:[0CH*4]
     MOV DS:OLD_VEC_OFF,BX
     MOV BX,WORD PTR ES:[0CH*4+2]
     MOV DS:OLD_VEC_SEG,BX
     CLI
     LEA BX,CS:INT_AD
     MOV ES:[0CH*4],BX
     PUSH CS
     POP  BX
     MOV ES:[0CH*4+2],BX
;разрешить все прерывания, в том числе и от COM-порта
     MOV AL,0
     OUT 21H,AL
;инициализация адаптера асинхронной связи
     MOV DX,ES:[400H]  ;порт COM1
     ADD DX,3
     IN  AL,DX
```

```
     AND AL,01111111B
     OUT DX,AL    ;на всякий случай сбросить 7-й бит
     SUB DX,2
     MOV AL,1
     OUT DX,AL    ;разрешить прерывания по получению данных
     STI
;ждем нажатия клавиши
     MOV AH,0
     INT 16H
;восстановим вектор и запретим прерывание от COM1
     CLI
     MOV BX,DS:OLD_VEC_OFF
     MOV WORD PTR ES: [0CH*4],BX
     MOV BX,DS:OLD_VEC_SEG
     MOV WORD PTR ES: [0CH*4+2],BX
     MOV AL,0
     OUT DX,AL
     STI
;выход в DOS
     MOV AH,4CH
     INT 21H
;процедура прерывания
INT_AD  PROC FAR
     PUSH DS
     PUSH AX
     PUSH DX
     PUSH ES
     MOV  AX,SEG STR
     MOV  DS,AX
     MOV  AX,0
     MOV  ES,AX
     MOV  DX,ES:[400H]
     IN   AL,DX    ;содержимое регистра данных
     CALL WRI_BYTE
;магическая последовательность
     MOV AL,20H
     OUT 20H,AL
;восстанавливаем регистры микропроцессора
     POP ES
     POP DX
     POP AX
     POP DS
     IRET
INT_AD  ENDP
```

```
;процедура вывода байта в десятичном виде
WRI_BYTE PROC
     PUSH AX
     PUSH DX
;вначале преобразование числа в строку ASCII
     XOR AH,AH
     MOV DL,10
     DIV DL
;предполагается, что DS указывает на сегмент данных (!)
;если не так, то следует об этом позаботиться
     ADD AH,48
     MOV DS:STR+2,AH
     XOR AH,AH
     DIV DL
     ADD AL,48
     ADD AH,48
     MOV DS:STR,AL
     MOV DS:STR+1,AH
;теперь печать
     LEA DX,DS:STR
     MOV AH, 9
     INT 21H
;инициализация строки для следующего вывода
     MOV BYTE PTR DS:STR,0
     MOV BYTE PTR DS:STR+1,0
     MOV BYTE PTR DS:STR+2,0
     POP DX
     POP AX
     RET
WRI_BYTE ENDP
CODE ENDS
     END BEG
```

Рис. 9.4. Простая иллюстрация взаимодействия с COM-портом по прерыванию.

Рассмотрим программу на Рис. 9.3 относительно работы с COM-портом. Обращаю Ваше внимание, что вектор прерывания 0CH взят потому, что программа работает с портом COM1. Если бы надо было работать с портом COM2, то потребовался бы вектор 0BH. Аналогично следует сказать об адресе портов. Базовый адрес порта COM1 находится в памяти по адресу 0:400H, COM2 - 0:402H. По базовому адресу находится регистр приема и передачи данных. Увеличив базовый адрес на единицу, мы получим адрес для регистра прерываний. Биты в этом регистре определяют, какие прерывания от адаптера будут разрешены. Мы разрешаем прерывание по получению данных (бит 0). Наконец, если мы увеличим базовый адрес на 3, то получим адрес для регистра управления. Наличие в этом регистре седьмого бита говорит о том, что два первых регистра

будут использоваться для задания скорости передачи данных. Поэтому мы сбрасываем этот бит, т.к. первые два регистра используются для других целей (см. выше).

VII.

В этом разделе приводится довольно большая законченная программа деления целых чисел. При запуске следует ввести вначале делимое, а после делитель. Числа вводятся в шестнадцатеричном виде. Данной программой следует заинтересоваться по следующим причинам:

а) Приводиться пример преобразования чисел из ASCII-формата в двоичный формат и обратно посредством специальных таблиц и команды XLAT

б) Перехватывается прерывание 0: если ввести нулевой делитель, то появится сообщение о недопустимости деления на 0 и произойдет выход из программы

```
DATA  SEGMENT
;таблицы перевода
;ASCII -> байт
TAB1  DB 48 DUP(O)
DB 0,1,2,3,4,5,6,7,8,9
DB 7 DUP(O)
DB 10,11,12,13,14,15
DB 26 DUP(0)
DB 10,11,12,13,14,15
DB 153 DUP(O)
;байт -> ASCII
TAB2  DB '0123456789ABCDEF'
;структура для ввода строки в 4 байта
MAX    DB 5  ; 4 байта строки + байт возврат каретки  (ODH)
     DB ?
STROKA DB 4 DUP(?)
     DB ?
MES    DB 'Деление на нуль недопустимо!',13,10,'$'
;здесь хранится старое значение вектора O
OLD_INT_OF  DW ?
OLD_INT_SEG DW ?
DATA ENDS
ST1  SEGMENT  STACK
     DW 50 DUP(?)
ST1  ENDS
CODE  SEGMENT
     ASSUME CS:CODE, DS:DATA, SS:ST1
/новая процедура прерывания, вызываемая при делении на O
INT_0  PROC
;сообщение о недопустимости нулевого делителя
     LEA   DX,MES
```

```
    MOV  AH, 9
    INT  21H
;восстановить вектор 0
    POP  ES
    MOV  AX,OLD_INT_SEG
    MOV  ES:[2],AX
    MOV  AX,OLD_INT_OF
    MOV  ES:[0],AX
;выйти из программы с кодом ошибки
    MOV  AX,4C01H
    INT  21H
INT_0 ENDP
BEGIN:
    MOV AX,DATA
    MOV DS,AX
;установка векторов
    XOR AX,AX
    MOV ES,AX
    PUSH ES
;вначале сохраним старое значение вектора
    MOV AX,ES:[0]
    MOV OLD_INT_OF,AX
    MOV AX,ES:[2]
    MOV OLD_INT_SEG,AX
    PUSH CS
    POP  AX
;теперь установим новый вектор, направив его на нашу процедуру
    MOV  ES:[2],AX
    LEA  AX,CS:INT_0
    MOV  ES:[0],AX
;ввод числа с клавиатуры
    LEA  SI,STROKA
;заполнить строку нулями
    CALL CLEAR_STR
    LEA DX,MAX
    MOV AH,0AH
;теперь ввод
    INT 21H
    MOV BL,MAX+1
;преобразование строки в число
    CALL SHIF
    CALL STR_NUM
;заполнить строку нулями
    CALL CLEAR_STR
```

```
     PUSH AX
     CALL ENT
;ввести делитель
     LEA  DX,MAX
     MOV AH,0AH
     INT 21H
     MOV BL,MAX+1
;преобразование строки в число
     CALL SHIF
     CALL STR_NUM
     LEA SI,STROKA
     MOV  BX,AX
     POP  AX
     XOR  DX, DX
;разделить
     DIV  BX
     PUSH DX
     CALL ENT
     CALL NUM_STR
;вывод результата деления
     CALL PRINT_STR
     POP  AX
     CALL NUM_STR
;вывод остатка
     CALL PRINT_STR
;восстановить вектор 0
     POP  ES
     MOV  AX,OLD_INT_SEG
     MOV  ES:[2],AX
     MOV  AX,OLD_INT_OF
     MOV  ES:[0],AX
_END:
     MOV AX,4C00H
     INT 21H
;область процедур
;преобразует 2 байта (16-ричные)  в число
/вход - DS:SI - строка
;выход  AX - число
STR_NUM PROC
     PUSH SI
     ADD SI,3
     MOV CX,2
LOO:
     XOR AH,AH
```

```
    MOV AL, [SI]
    LEA BX,TAB1
    XLATB
    MOV DI,AX
    MOV AL,[SI]-1
    LEA BX,TAB1
    XLATB
    MOV DL,16
    MUL DL
    ADD AX,DI
    PUSH AX
    SUB  SI,2
    LOOP LOO
    POP  AX
    MOV  CL,8
    SHL  AX,CL
    POP  BX
    ADD  AX,BX
    POP  SI
    RETN
STR_NUM ENDP
;преобразует число в строку
;число находится в AX
;на строку указывает DS:SI
NUM_STR PROC
    PUSH AX
    MOV  CL,8
    SHR  AX,CL
    MOV  BL,16
    DIV  BL
    LEA BX,TAB2
    XLATB
    MOV  [SI],AL
    MOV AL,AH
    LEA BX,TAB2
    XLATB
    MOV  [SI]+1,AL
    POP AX
    AND AX,00FFH
    MOV  BL,16
    DIV  BL
    LEA  BX,TAB2
    XLATB
    MOV  [SI]+2,AL
```

```
        MOV AL,AH
        LEA BX,TAB2
        XLATB
        MOV [SI+3],AL
        RETN
NUM_STR ENDP
;перевод строки
ENT PROC
        PUSH AX
        PUSH DX
        MOV DL,13
        MOV AH,2
        INT 21H
        MOV DL,10
        MOV AH,2
        INT 21H
        POP DX
        POP AX
        RETN
ENT ENDP
;заполнение строки символом '0'
;DS:SI - на строку
CLEAR_STR PROC
        PUSH SI
        PUSH AX
        MOV CX, 4
LOO2:
        MOV BYTE PTR [SI],'0'
        INC SI
        LOOP L002
        POP AX
        POP SI
        RETN
CLEAR_STR ENDP
;сдвиг строки
;в BL длина строки
;DS:SI - адрес строки
;преобразование типа 78 -> 0078
SHIF PROC
        CMP BL,0
        JNZ NO_ZER
        RETN
NO_ZER:
        PUSH SI
```

```
        XOR BH,BH
        MOV DI,SI
        ADD DI,3
LOO1:
        MOV AL,[SI][BX]-1
        MOV BYTE PTR [SI][BX]-1,'0'
        MOV [DI],AL
        DEC DI
        DEC BX
        JNZ L001
        POP SI
        RETN
SHIF ENDP
;печать строки
;DS:SI - на строку
PRINT_STR PROC
        MOV CX,4
        MOV AH,2
LOO4:
        MOV DL, [SI]
        INT 21H
        INC SI
        LOOP LOO4
        CALL ENT
        RETN
PRINT_STR ENDP
CODE ENDS
        END BEGIN
```

Рис. 9.5. Программа деления нацело четырехзначных чисел, представленных в шестнадцатеричном виде.

После того, как Вы познакомились с текстом программы и проверили, как она работает, прочтите и пояснения к ней.

1. В программе заменяется стандартный обработчик деления на 0 (вектор 0) на нашу процедуру. При нулевом делителе появляется сообщение о недопустимости деления на 0. Прямо из этой процедуры мы выходим в операционную систему. Для такой программы это естественно, но есть и проблема. Дело в том, что если для компьютеров на базе микропроцессоров 8088/8086 в стек помещался адрес следующей команды (следующей за DIV или IDIV), то для следующих поколений микропроцессоров в стек помещается адрес самой команды деления. Ставить в конце процедуры обработки команду IRET было бы бессмысленно: возник бы бесконечный цикл вызова процедуры. Мы пошли по самому простому пути и сразу передаем управление операционной системе. Более сложные пути должны предполагать переходы из процедуры в те или иные точки программы (с освобождением стека, естественно).

2. Структура, начинающаяся со слова MAX, служит и для ввода строк и для их вывода (см. функция DOS OAH). Процедуры CLEAR_STR и SHIF служат для представления вводимых чисел в удобном для преобразования виде. Первая заполняет строку символами '0', а вторая осуществляет преобразования типа 123 -> 0123, 2->0002 и т.п. Процедуры STR_NUM и NUM_STR осуществляют преобразования строки в число и обратно.

3. Замечу в заключение, что данная программа является прекрасной иллюстрацией использование такой команды как, XLAT.

VIII.

В Приложении 9, в разделе, посвященном таймеру, приведена программа с процедурой, осуществляющей задержку во времени. Задержки в программах применяются довольно часто. Как правило, для этой цели используют какой-либо циклический алгоритм. Однако такая задержка является машинно-зависимой. В Приложении 9 программа работает правильно независимо от производительности процессора. Алгоритм основан на непрерывном опросе таймера. Здесь предлагается другой вариант процедуры задержки с использованием прерывания 1CH.

```
CODE  SEGMENT
      ASSUME CS:CODE, DS:CODE
      ORG  100H
BEGIN:
      MOV AH,9
      LEA  DX,TEXT1
      INT 21H
;количество секунд задержки
      MOV CX,10
      CALL  TIME
      MOV AH,9
      LEA  DX,TEXT2
      INT 21H
      RET
TIME  PROC
;установить вектор прерывания
      XOR AX,AX
      MOV ES,AX
;вначале сохранить старый вектор
      MOV  AX,ES:[1CH*4]
      MOV  INTOFF,AX
      MOV  AX,ES:[1CH*4+2]
      MOV  INTSEG,AX
;теперь установить новый вектор
      CLI
      LEA AX,INT1C
```

```
     MOV ES:[1CH*4],AX
     PUSH CS
     POP  AX
     MOV  ES:[1CH*4+2],AX
;установить счетчики
     MOV COUNT,CX
     MOV AH,2CH
     INT 21H
     MOV SEC,DH
     STI
;цикл проверки счетчика
LOO:
     CMP CS:COUNT,0
     JNZ LOO          ;продолжить, если счетчик не нулевой
;восстановить вектор
     CLI
     MOV  AX,INTOFF
     MOV  ES:[1CH*4],AX
     MOV  AX,INTSEG
     MOV  ES:[1CH*4+2],AX
     STI
     RETN
TIME ENDP
INT1C PROC
     MOV AH,2CH
     INT 21H
     CMP CS:SEC,DH
     JZ CONT
     DEC  CS:COUNT
     MOV CS:SEC,DH
CONT:
     IRET
INT1C ENDP
;хранится количество секунд текущего времени
SEC DB ?
;счетчик интервала
COUNT DW ?
TEXT1 DB 'Ждите .. 10 с.',13,10,'$'
TEXT2 DB 'Конец',13,10,'$'
INTSEG DW ?
INTOFF DW ?
CODE ENDS
     END BEGIN
```

Рис. 9.6. Демонстрация процедуры задержки.

Принцип работы программы, приведенной на Рис. 9.6, следующий: процедура, вызываемая через прерывание 1CH, приблизительно 18 раз в секунду проверяет, не изменилось ли значение секунд системных часов. Если значение изменилось, то счетчик секунд уменьшается. Параллельно этому ведется непрерывный опрос счетчика секунд. Выход из цикла происходит, когда счетчик становится равным нулю. Особо отметьте, что весь алгоритм построен на том факте, что длительность всех выполняемых команд намного меньше секунды. В противном случае нам пришлось бы проверять, на сколько секунд изменился показатель системных часов.

IX.

Как я и обещал, возвратимся снова к проблеме обработки критической ошибки, т.е. к прерыванию 24H.

Программа, приведенная на Рис. 9.7 не является законченным примером обработки критической ошибки. Она представляет, если хотите, маленькую лабораторную работу для уяснения всех проблем, которые встанут перед Вами, если Вы всерьез захотите, чтобы программы корректно обрабатывали проблемы, возникающие при работе с гибкими дисками.

```
DATA SEGMENT
PATH DB "A:\FILE",0    ;имя открываемого файла HANDLE DW ?
;описатель открытого файла
HANDLE DW ?
TEXT1 DB "Жду нажатия клавиши",13,10,"$"
TEXT2 DB "Файл создан",13,10,"$"
TEXT3 DB "Произошла ошибка",13,10,"$"
DATA ENDS
STA SEGMENT STACK
    DB 500 DUP(0)
STA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:STA
BEGIN:
    MOV AX,DATA
    MOV DS,AX
;переустановить вектор 24H
    MOV AX,0
    MOV ES,AX
    LEA AX,CS:INT24
    MOV ES:[24H*4],AX
    MOV ES: [24H*4+2] ,CS
INP:
    LEA DX,TEXT1
    CALL TEXTOUT
    CALL INPUT          ;ждем нажатие клавиши
```

```
     CMP  AL,27        ;проверка на нажатие ESC
     JZ   _END
;открыть файл
     MOV  AX,3C00H
     LEA  DX,PATH
     MOV  CX,0
     INT  21H
     JNC  DAL2
     LEA  DX,TEXT3
     CALL  TEXTOUT
     JMP SHORT INP          ;повторить операцию открытия
DAL2:
     MOV  DS:HANDLE,AX
;записать в файл
     LEA   DX,TEXT2
     CALL TEXTOUT
INP1:
     LEA DX,TEXT1
     CALL TEXTOUT
     CALL INPUT
     LEA  DX,TEXT2
;производим операцию записи
     MOV  BX,DS:HANDLE
     MOV  CX,11
     MOV  AX,4000H
     INT  21H
     JNC   DAL1
     LEA  DX,TEXT3
     CALL TEXTOUT
     JMP  SHORT  INP1       ;повторить операцию записи
DAL1:
;закрыть файл
INP2:
     LEA  DX,TEXT1
     CALL TEXTOUT
     MOV  BX,HANDLE
     MOV  AH,3EH
     INT  21H
     JNC  _END
     LEA  DX,TEXT3
     CALL TEXTOUT
     JMP SHORT INP1         ;повторить операцию закрытия
_END:
     MOV  AX,4C00H
```

```
     INT 21H
INPUT PROC
     MOV AH,0
     INT 16H
     RET
INPUT ENDP
;обработчик прерывания 24H
INT24 PROC
     MOV AL,1
     IRET
INT24 ENDP
;вывод текстовой строки
TEXTOUT PROC
     MOV AH,9
     INT 21H
     RET
TEXTOUT ENDP
CODE ENDS
     END BEGIN
```

Рис. 9.7. Обработка критических ошибок.

Главное, что Вы должны для себя уяснить, работая с данной программой, это то, как обрабатываются операции открытия файла, записи в файл и закрытия файла в случае возникновения критической ошибки. В нашем случае AL=1 и все операции обрабатываются одинаково, т.е. повторяются до тех пор, пока не перестанет происходить ошибка. При других значениях AL операции обрабатываются по-разному. Вы это проверите сами, замечу только, что логика разработчиков вполне понятна. Если, к примеру, не удалось открыть файл, то ситуация вполне однозначна: данные не сохранены или не прочитаны. Если же не удалось записать данные, то здесь возникает некоторая неопределенность: не ясно, что с файлом. Так что разработчиков понять можно.

С версии 4.0 MS DOS, однако, появилась функция 6CH, позволяющая открывать файлы так, что критическая ошибка не вызывает прерывание. О наличии любой ошибки можно, естественно, судить по флагу переноса. Расширенный код ошибки получается, как и обычно, через функцию 59H. Остается открытым вопрос: зачем вообще нужно было выделять таким образом критическую ошибку?

В заключение замечу, что при входе в процедуру обработки критической ошибки стек содержит всю нужную информацию, чтобы самостоятельно можно было обработать возникшую ситуацию. Вот структура стека:

1. Адрес возврата в MS DOS (3 слова: IP,CS,Flags). Выполняя IRET, мы как раз используем эти данные.
2. Значения регистров перед вызовом int 21H: AX,BX,CX,DX,SI,DI,BP,DS,ES.
3. Адрес возврата к команде int 21H в вашей программе (3 слова: IP,CS,Flags).

X. Перехват прерываний.

В этом разделе будет представлен модуль, который может быть использован как в программах на языке ассемблера, так и на языках высокого уровня. Вызов процедуры INI_KB должен производиться в начале программы. Перед выходом из программы должна быть вызвана процедура RE_KB. После выполнения процедуры INI_KB перестают работать клавиши PAUSE, PRTSC, сочетания клавиш Ctrl Break, Ctrl C, Ctrl Alt Del. Все остальное работает по-прежнему. Блокирование клавиш производится независимоттого,какиефункциивыполняетпрограмма.

```
CODE  SEGMENT
     ASSUME CS:CODE
     PUBLIC INI_KB, RE_KB
;старые векторы
INT_05O DW ?
INT_05S DW ?
INT_1BO DW ?
INT_1BS DW ?
INT_09O DW ?
INT_09S DW ?
;--------------
;новые процедуры обработки прерываний
;-обработка Prtsc
_05 PROC
     IRET
_05 ENDP
;-обработка Ctrl Break
_1B PROC
     IRET
_1B ENDP
SCAN DB ?
;-обработка клавиатурного прерывания
_09 PROC
     PUSH AX
     IN   AL,60H
     MOV  CS:SCAN,AL
     PUSH ES
     XOR  AX,AX
     MOV  ES,AX
;обработка Ctrl Alt Del
     TEST BYTE PTR ES: [417H],00000100B
     JZ   PROD
     TEST BYTE PTR ES : [417H],00001000B
     JZ   PROD
```

```
     AND  BYTE PTR ES: [417H], 11110111B
     POP  ES
     JMP  TO_INT9
;-----
PROD:
;обработка Ctrl C
     CMP CS:SCAN, 46
     JNZ PRODI
     TEST BYTE PTR ES: [417H],00000100B
     JZ PRODI
     POP ES
     JMP SHORT EMUL
PROD1:
     POP  ES
TO_INT9:
     PUSHF
     CALL DWORD PTR CS:INT_09O
     PUSH ES
     XOR  AX,AX
     MOV  ES,AX
;отключить клавишу PAUSE
     AND  BYTE PTR ES: [418H],11110111B
     POP  ES
     POP  AX
     IRET
EMUL:
     IN AL,61H
     MOV AH,AL
     OR  AL,80H
     OUT 61H,AL
     MOV AL,AH
     OUT 61H,AL
     MOV AL,20H
     OUT 20H,AL
     POP AX
     IRET
_09 ENDP
;--------------
;инициализировать обработку
INI_KB PROC FAR
     CLI
     PUSH AX
     PUSH ES
     XOR  AX, AX
```

```
     MOV  ES,AX
     MOV  AX,ES:[05H*4]
     MOV  CS:INT_05O,AX
     MOV  AX,ES:[05H*4+2]
     MOV  CS:INT_05S,AX
     LEA  AX,CS:_05
     MOV  ES:[05H*4],AX
     MOV  ES:[05H*4+2],CS
     ;--
     MOV  AX,ES:[1BH*4]
     MOV  CS:INT_1BO,AX
     MOV  AX,ES:[1BH*4+2]
     MOV  CS:INT_1BS,AX
     LEA  AX,CS:_1B
     MOV  ES:[1BH*4],AX
     MOV  ES:[1BH*4+2],CS
     ;--
     MOV  AX,ES:[09H*4]
     MOV  CS:INT_09O,AX
     MOV  AX,ES:[09H*4+2]
     MOV  CS:INT_09S,AX
     LEA  AX,CS:_09
     MOV  ES:[09H*4],AX
     MOV  ES:[09H*4+2],CS
     POP  ES
     POP  AX
     STI
     RETF
INI_KB ENDP
;восстановить обработку
RE_KB PROC FAR
     CLI
     PUSH AX
     PUSH AX
     XOR  AX,AX
     MOV  ES,AX
;восстанавливаем векторы прерываний
     MOV  AX,CS:INT_05O
     MOV  ES:[05H*4],AX
     MOV  AX,CS:INT_05S
     MOV  ES:[05H*4+2],AX
     ;--
     MOV  AX,CS:INT_1BO
     MOV  ES:[1BH*4],AX
```

```
      MOV  AX,CS:INT_1BS
      MOV  ES:[1BH*4+2],AX
      ;--
      MOV  AX,CS:INT_09O
      MOV  ES: [09H*4],AX
      MOV  AX,CS:INT_09S
      MOV  ES: [09H*4+2],AX
      ;--
      POP  AX
      POP  AX
      STI
      RETF
RE_KB ENDP
CODE ENDS
      END
```

Рис. 9.8. Пример обработки прерываний.

Глава 10. Введение в графическое программирование.

Когда б вы знали, из какого сора
Растут стихи, не ведая стыда.
Анна Ахматова

В данной главе на примере всего лишь одного графического режима, который существовал еще для EGA-адаптеров, будут рассмотрены основные принципы создания графических изображений на экране. На более высоком уровне программирование видеосистем будет рассмотрено в главе 27.

Операционная система MS DOS в отличие от Windows (см. главы 24, 25), к сожалению, не поддерживает доступ к графическим возможностям компьютера. В BIOS есть не слишком эффективный графический интерфейс - подфункции прерывания ЮН. Вещь в значительной степени прискорбная, т.к. приходится программировать видеоадаптер самому. Это приводит к непереносимости программ на различных типах компьютеров. Кроме того, программировать видеоадаптер - достаточно сложная задача. Здесь есть две стороны. Мне, как программисту, нравятся сложные задачи, программирование адаптеров, нестандартные подходы. Однако если Вы пишите коммерческую программу, то решающее слово здесь за потребителем. Ему же необходимы: быстрота программирования, удобство пользования и совместимость. На мой взгляд, однако, плох программист, который мыслит как тот, кому предназначены его программы. Пользователь должен возвращать нас на землю, по своей же воле не стоит этого делать.

Для языков высокого уровня существуют графические библиотеки, которые решают большинство задач, необходимых для успешной работы с графическим экраном. Кроме того, стандартные библиотеки, как правило, рассчитаны на работу с разными адаптерами и в разных графических режимах. Однако никакая библиотека не может охватить всех возможностей графических адаптеров (особенно VGA и SVGA). Кроме того, часто требуются специальные эффекты, добиться которых можно лишь, работая с адаптером напрямую.

Прежде чем перейти к программированию на низком уровне, рассмотрим графические возможности BIOS, которые будут касаться только VGA-адаптеров (см. [8]). В главе 7 мы коснулись вопроса определения типа видеоадаптера.

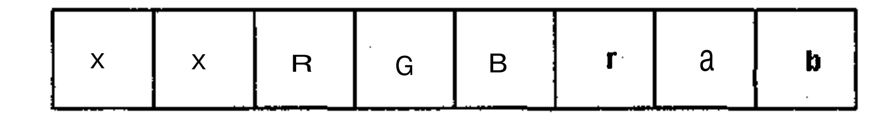
I.

Ниже перечисляются основные графические функции прерывания 1OH.

Установка режима.
Вход:
AH - O, AL - номер режима.

Режимы для EGA и VGA-адаптеров изменяются в промежутке 14-19 (режимы 17-19 для VGA только). Для определенности в дальнейшем мы будем рассматривать только режим с номером 16 - разрешение 640*350, 16 цветов.

Доступ к регистрам палитры.
АН - 10Н
AL - 0 - изменить регистры палитры.
BL - номер регистра,
ВН - цвет (6 бит). AL - 1 - изменить регистры бордюра. ВН - регистр бордюра.
AL - 2 - изменить регистры палитры и бордюра.
ES:BX - 17 байт (регистры палитры 16, 17-й бордюра).
AL - 3 - интенсивность.
BL - 0 интенсивный фон (16 - цветов).
BL - 1 мерцание (8 цветов + мерцание пер. плана).
Структура байта палитры имеет следующий вид:



Последние два бита в байте не используются. Биты Rr определяют интенсивность красного цвета (red), биты Gg - интенсивность зеленого цвета (green), биты Bb - интенсивность синего цвета (blue). Таким образом, любой цвет получается смешением трех чистых цветов, причем каждый цвет представлен с определенной интенсивностью (от 0 до 3).

Поставить точку.
АН - 0СН
ВН - номер видеостраницы.
DX - строка.
СХ - столбец.
AL - значение цвета.
Читать точку.
АН - 0DH.
Регистры работают аналогично предыдущему случаю, в AL возвращается цвет.
Выбрать активную страницу (переключение страниц).
АН - 5
AL - номер активной страницы (для рассматриваемого нами режима их всего 2).

На Рис. 10.1 демонстрируются возможности функций BIOS. При запуске Вы легко убедитесь, что вывод точки работает чрезвычайно медленно. То же можно сказать о выводе символа, поскольку символ выводится как набор точек. Однако такая функция, как смена активной страницы, работает практически мгновенно. Этот механизм часто

используют в играх и мультипликации. Замечу, что в графическом режиме нет курсора в обычном понимании (мы его не видим). Однако именно в его позицию выводится символ. Причем позицию графического курсора можно изменить при помощи обычных функций установки курсора. Использование русского текста возможно лишь в том случае, если это обеспечивает используемый драйвер экрана. Как правило, хорошие графические библиотеки для языков высокого уровня содержат наборы шрифтов, а также функции управления их выводом.

```
CODE SEGMENT
       ASSUME CS:CODE
       ORG 100H
BEGIN:
;в графический режим
       MOV AX,0010H
       INT 10H
;вывод прямоугольника по точкам
       MOV BH,0
       MOV AH,0CH
       MOV SI,120      ;горизонталь
       MOV DI,50       ;вертикаль
       MOV AL,13       ;цвет прямоугольника
       MOV DX,10       ;строка
       MOV CX,30       ;колонка
LO:
       INT 10H
       INC CX
       DEC SI
       JNZ LO
       INC DX
       MOV CX,30
       MOV SI,120
       DEC DI
       JNZ LO
;изменение палитры
       MOV CX, 64
       MOV AX,1000H
       MOV BL,13
       MOV BH,0
LOO:
       PUSH AX
;ждем нажатия клавиши
       XOR  AH,AH
       INT  16H
       CMP  AL,27
```

```
        POP   AX
        JZ    OUT_C
        INT   ЮН
        INC   BH
        LOOP  LOO
;выводим символ на страницу О
OUT_C:
        XOR BH,BH
        MOV DH,15
        MOV DL,35
        MOV AH,2
        INT ЮН
        MOV AL,128    ;А - русское
        MOV AH,0AH
        MOV CX,1
        INT 10H
;теперь на страницу 1
        MOV BH,1
        MOV DH,15
        MOV DL,35
        MOV AH,2
        INT 10H
        MOV AL,129    ;Б - русское
        MOV AH,0AH
        MOV CX,1
        INT 10H
        XOR AH,AH
        INT 16H
;выбираем активную страницу
        MOV AH,05
        MOV AL,1
        INT ЮН
        XOR AH,AH
        INT 16H
;в текстовый режим
        MOV AX,0002H
        INT 10H
; выходим в DOS
        RET
CODE ENDS
        END BEGIN
```

Рис. 10.1. Демонстрация графических возможностей прерывания ЮН.

II.

Для того чтобы овладеть всеми графическими возможностями адаптера, нам не избежать подробного рассмотрения всех его регистров, а также режимов чтения и записи.

При использовании графических режимов VGA адрес видеобуфера расположен, начиная с адреса A000:0000. Однако адресное пространство видеобуфера составляет всего 64 К. Реальный же размер видеобуфера составляет до 256 К. Для хранения одной страницы экрана в режиме 16 требуется 640*350/2= 122500 байт. Ясно, что адресного пространства явно не хватает и для чтения и записи в видеобуфере приходится делать некоторые ухищрения.

Реально содержимое видеопамяти формируется содержимым четырех битовых плоскостей. Каждая битовая плоскость отвечает за свой цвет. Байт битовой плоскости соответствует восьми пикселям экрана. Таким образом, цвет каждой точки формируется четырьмя битами, и мы получаем 16 цветов. Четырем битовым плоскостям соответствуют четыре регистра-защелки. При чтении из видеопамяти регистры-защелки заполняются соответствующим содержимым - цветами восьми пикселей. При записи в видеопамять регистры-защелки участвуют в формировании цветов восьми пикселей. Для микропроцессора непосредственный доступ в регистры-защелки закрыт. Однако при записи необходимо учитывать содержимое этих регистров.

Из сказанного выше следует, что для формирования экранной области требуется 122500/4=30625 байт адресного пространства. 64К адресного пространства хватит для двух страниц видеопамяти.

Рассмотрим основные регистры адаптера и их краткую характеристику (в главе 27 будет дан полный справочник видеорегистров).

Регистр 0 (установки/сброса). Позволяет установить или сбросить значение байта в четырех битовых плоскостях.

Регистр 1 (разрешение установки/сброса). Управляет работой регистра установки/сброса. Записывая в какой-либо бит этого регистра единицу, мы разрешаем использование соответствующего бита регистра установки сброса.

Регистр 2 (регистр сравнения). Определяет цвет для режима чтения 1 (см. ниже).

Регистр 3 (регистр циклического сдвига данных/регистр выбора функций). В этом регистре независимо друг от друга работают две битовые группы: группа из трех бит 0..2 определяет циклический сдвиг байта, передаваемого из микропроцессора в видеопамять; сдвиг осуществляется слева направо. Группа из двух битов 3 и 4 определяет логическую функцию (операцию), выполняемую при передаче в видеопамять над байтом данных микропроцессора и содержимым буфера данных видеопамяти. Если биты равны 0, то байт передается без изменений, если бит 3 равен 1, а бит 4 равен 0, то выполняется операция логического "И", если бит 4 равен 1, а бит 3 равен 0, то выполняется операция логического "ИЛИ", если оба бита равны 0, то выполняется операция исключающего "ИЛИ".

Регистр 4 (регистр выбора битовой плоскости). Используется для выбора битовой плоскости в операциях чтения из видеопамяти.

Регистр 5 (регистр режима). Предназначен для выбора режима чтения/записи. Биты 0-1 устанавливают режим записи (режим 3 только для VGA и выше). Бит 3 устанавливает режим чтения (0 или 1).

Регистр 6 (регистр добавочных функций). Бит 0 равен 0 при работе в алфавитно-цифровом режиме. Бит 1 - выбор отображаемой на экран страницы. Биты 2-3 определяют, какие адреса памяти отображаются на экран:

БИТ 3	БИТ 2	Адрес видеобуфера
0	0	0A0000H-0BFFFFH
0	1	0A0000H-0AFFFFH
1	0	0B0000H-0B7FFFH
1	1	0b8000H-0BFFFFH

Регистр 7 (регистр запрещения чтения цвета или регистр фильтрации). В режиме чтения 1 запрещает передачу цвета или набор цветов в микропроцессор.

Регистр 8 (регистр битовой маски). Если бит в этом регистре установлен в 1, то соответствующий пиксель байта видеобуфера модифицируется операцией записи. Этот регистр работает во всех режимах записи.

Перечисленные регистры выбираются записью его номера в порт 3CEH. После этого значение, которое мы собираемся поместить в регистр, помещается в порт 3CFH.

Нам понадобится еще один важный регистр адаптера - регистр маски карты. Этот регистр имеет адрес порта 3C5H. Перед посылкой данных в порт 3C4H следует послать число 2 (произвести индексацию). Данный регистр позволяет маскировать отдельные битовые плоскости и тем самым задавать цвет выводимой точки (чаще используется в режиме записи 0).

Ниже (Рис. 10.2-10.7) представлены схемы всех режимов чтения-записи адаптера EGA. Рассмотрим каждый рисунок.

Режим чтения 0. Данный режим устанавливается по умолчанию во время загрузки машины. Процессор получает байт, содержащийся в одной из битовых плоскостей. Номер битовый плоскости помещается в регистр выбора битовой плоскости.

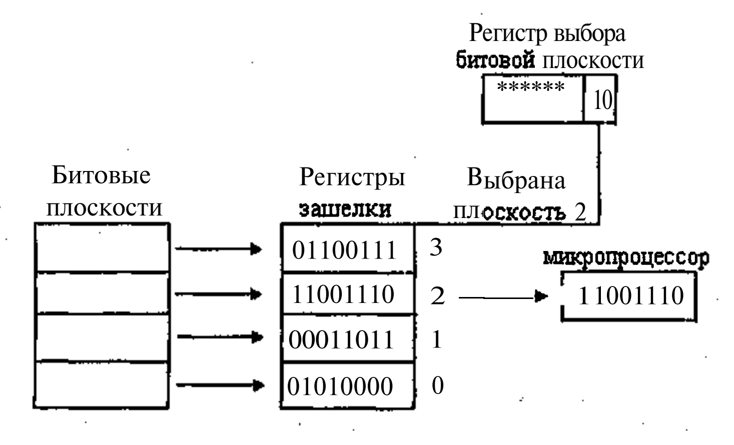


Рис. 10.2. Режим чтения 0.

Режим чтения 1. В этом режиме результат сравнения содержимого регистра сравнения цвета с цветом каждого из восьми пикселей передается микропроцессору. Если сравнение показало идентичность, то соответствующий бит будет равен 1. Перед сравнением над цветом каждого пикселя производится логическая операция "и" с содержимым регистра маскирования битовых плоскостей.

Режим записи 0(а). Если в регистре разрешения установки/сброса содержится OFH, то байт в видеопамять передается из регистра установки/сброса. При этом меняются битовые плоскости только у разрешенных пикселей (регистр битовой маски). При этом берется во внимание содержимое регистра сдвига: биты сдвига 0-2, биты операции 3-4. На Рис. 10.4 все эти биты равны 0, поэтому ни сдвига, ни операции не производится.

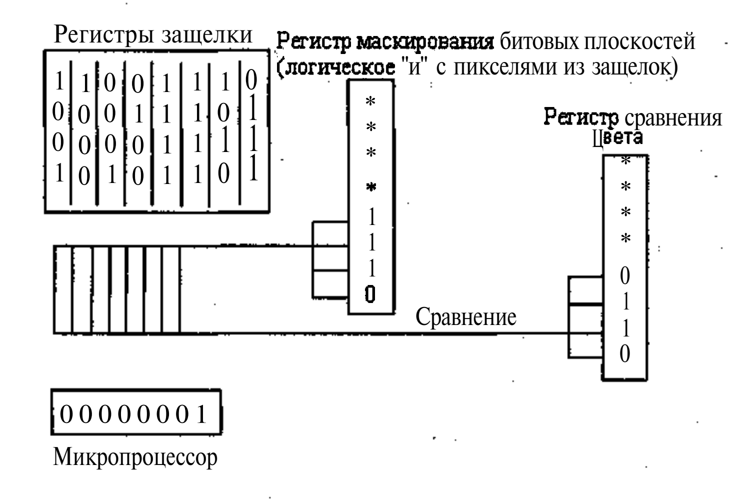


Рис. 10.3. Режим чтения 1.

Режим записи 0(б). В данном режиме байт передает микропроцессор. В нашем случае передается 11111111B. Это значит, что для каждого пикселя передается число 15 (1111). Однако регистр маски карты маскирует некоторые биты. В результате к пикселям направляется число 1001B. Разумеется, реально это число доходит лишь до пикселей, разрешенных регистром битовой маски. Наконец, не забудьте еще о регистре сдвига.

Режим 0(б) позволяет формировать изображение на экране двумя способами:

1. Цвет для каждой точки помещается в регистр маски карты. Точка определяется содержимым регистра битовой маски. Т.е. изображение есть последовательность атрибутов точек. Пример постановки точки этим способом будет дан в главе 27.

2. Цвет определяется сразу для восьми точек последовательностью четырех байт. При этом регистр битовой маски должен содержать FFH, т.е. разрешается запись для всех точек. Формирование цвета восьми точек осуществляется последовательной записью в четыре битовые плоскости. Номер битовой плоскости определяется содержимым регистра разрешения записи битовых плоскостей (1, 2, 4, 8 соответственно для плоскостей 0, 1, 2, 3). Легко видеть, что для формирования изображений второй способ предпочтительнее, тогда как первый способ удобнее для задания цвета отдельных точек.

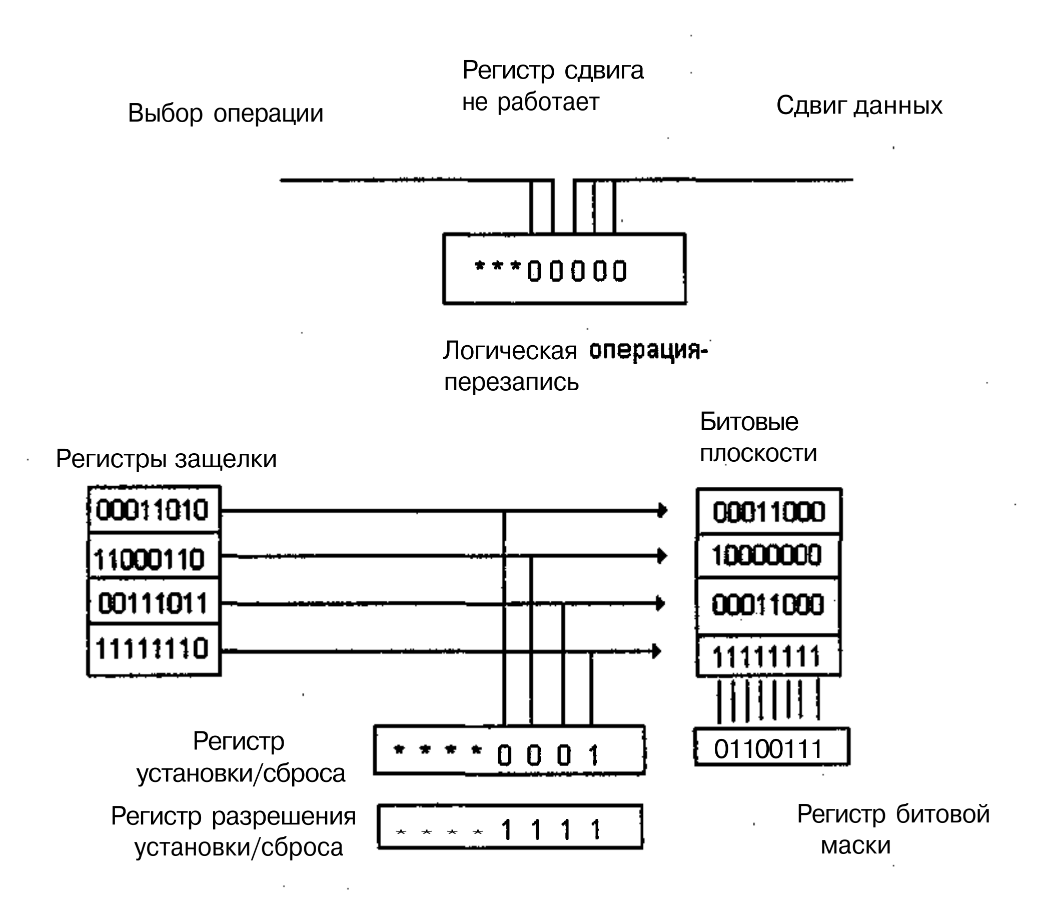


Рис.10.4. Режим записи 0(а).

Режим записи 1. При этом режиме данные передаются из регистров защелки непосредственно в видеопамять. Режим хорош для передачи данных из одной области видеопамяти в другую.

Режим записи 2. Цвет точки передается непосредственно процессором: биты 0-3 (с учетом регистра сдвига, разумеется).

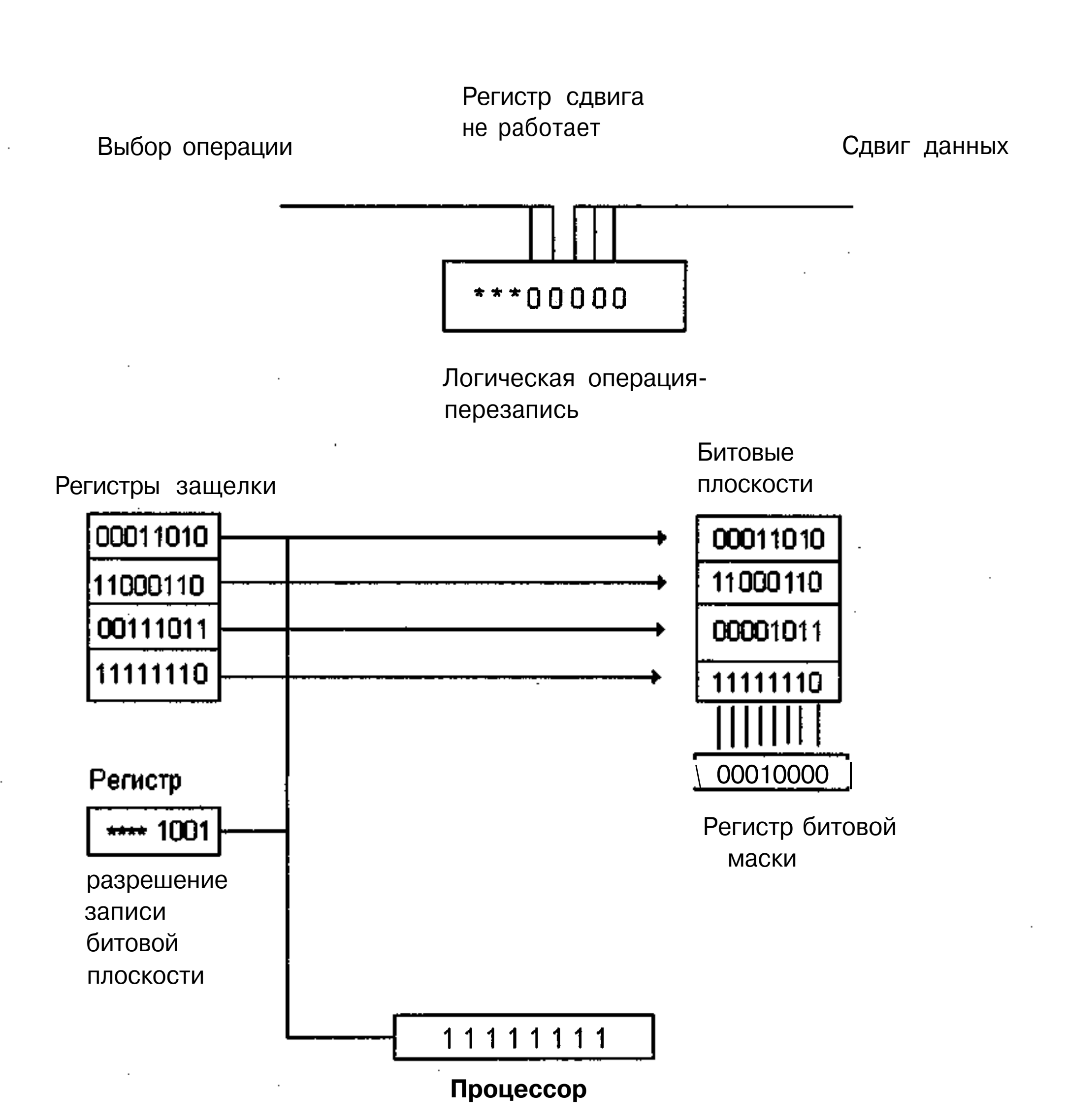


Рис. 10.5. Режим записи 0(б). Ставим одну точку. Цвет 9 (1001).

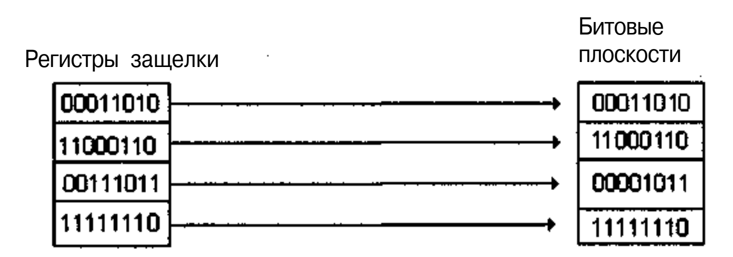


Рис. 10.6. Режим записи 1.

Регистр сдвига не работает

Выбор операции

Сдвиг данных

***00000

Логическая операция-перезапись

Битовые плоскости

Регистры защелки

01011010

11000110

00111011

01111110

01011010

11000110

00010101

01111110

00010000

Регистр битовой маски

*** 1 0 1 1

Процессор

Рис. 10.7. Режим записи 2. Ставим точку. Цвет 11 (1011).

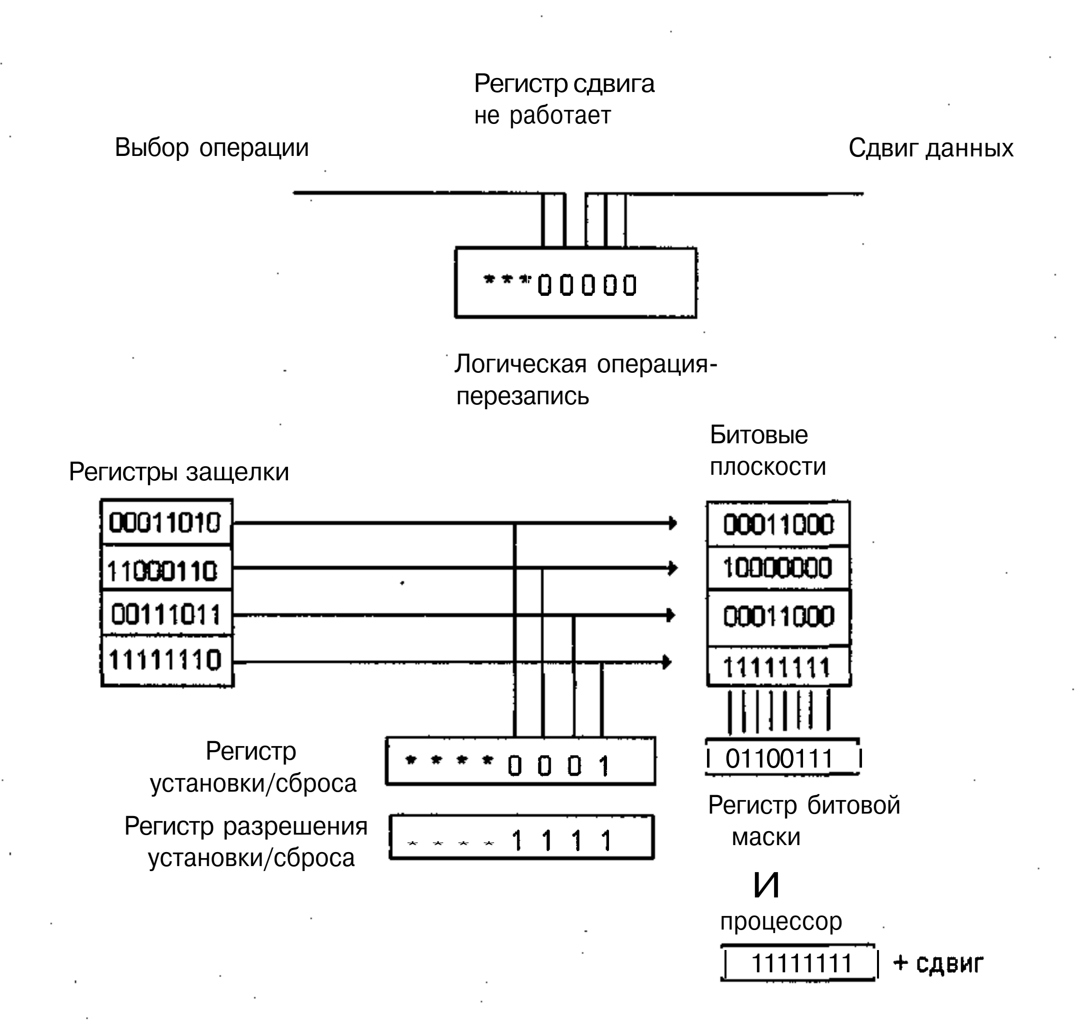


Рис. 10.4. Режим записи 3.

Режим записи 3. Работает только для адаптеров VGA и выше. Похож на режим записи 0 (а). Но здесь участвует и байт от процессора. Предварительно над этим байтом производится сдвиг согласно регистру сдвига. Далее над ним и байтом регистра битовой маски производится операция "И". Результирующий байт и будет байтом для маскирования пикселей. Кроме того, регистр разрешения установки/сброса больше не влияет на то, какие биты регистра установки/сброса участвуют в формировании битовых плоскостей.

Пример использования такого типа записи будет дан в главе 27.

В следующем примере (Рис. 10.7) на экран выводится точка цвета DH. Используется режим записи 0(б). Вот моменты, на которые следует обратить внимание:

1. Вывод в регистр адаптера производится всего тремя командами. Например, для установки режима чтения-записи:

```
MOV  DX,3CEH
MOV AX,0005H
OUT  DX,AX
```

Более привычны для Вас были бы команды:

```
MOV  DX,3CEH
MOV AL,5
OUT  DX,AL
INC  DX
MOV AL,0
OUT  DX,AL
```

Первый способ более краток и быстр и потому предпочтительнее второго.

2. Перед записью мы читаем из видеобуфера для того, чтобы содержимое регистров защелки соответствовало содержимому видеобуфера. В противном случае при выполнении операции записи может измениться цвет как раз у тех точек, которые мы замаскировали в регистре битовой маски.
3. В нашем примере мы выводим точку на черном фоне. Т.е. содержимое битовых плоскостей было нулевым. Если бы это было не так, то нам перед посылкой нужного цвета следовало бы вначале обнулить битовые плоскости.

```
CODE   SEGMENT
       ASSUME CS:CODE
       ORG 100H
BEGIN:
;в графический режим
       MOV AX,0010H
       INT 10H;выводим точку, режим 0;выбор режима
       MOV  DX,3CEH
       MOV AX,0005H        ;режим записи-чтения 0
       OUT DX,AX
;регистр сдвига
       MOV AX,0003H        ;запись без изменения
       OUT DX,AX
;выбор точки в байте
       MOV AX,8008H        ;первая точка байта (слева направо)
       OUT DX,AX
;выбор цвета
       MOV DX,3C4H
       MOV AX,0D02H         ;второй регистр, цвет DH
       OUT DX,AX
```

```
;устанавливаем указатель на нужный байт
        MOV AX,0A000H
        MOV ES,AX
        MOV BX,80*20+2               ;двадцатая строка
        MOV AL,BYTE PTR ES:[BX]      ;вначале читаем
        MOV BYTE PTR ES:[BX],0FFH
;ждем нажатия клавиши
        MOV AH,0
        INT 16H
;в текстовый режим
        MOV AX,0002H
        INT 10H
;выходим в DOS
        RET
CODE ENDS
        END BEGIN
```

Рис. 10.8, Вывод точки на экран в режиме записи ***0.***

Обратимся теперь к Рис. **10.9,** который демонстрирует режим записи 2. Видно, что данный режим проще предыдущего, т.к. не используется регистр маски карты.

```
CODE SEGMENT
        ASSUME CS:CODE
        ORG 100HBEGIN:;в графический режим
        MOV AX,0010H
        INT 10H
;выводим точку, режим 2
;выбор режима
        MOV DX,3CEH
        MOV AX,0205H
        OUT DX,AX
/регистр сдвига
        MOV AX,0003H
        OUT DX,AX
;выбор точки в байте
        MOV AX,8008H
        OUT DX,AX
;устанавливаем указатель на нужный байт
        MOV AX,0A000H
        MOV ES,AX
        MOV BX,80*20+2
        MOV AL,BYTE PTR ES:[BX]
        MOV AL,13
        MOV BYTE PTR ES:[BX],AL
```

```
        MOV  AH,0
        INT  16H
;в текстовый режим
        MOV  AX,0002H
        INT  10H
;выходим в DOS
        RET
CODE  ENDS
        END  BEGIN
```

Рис. 10.9. Вывод точки на экран в режиме записи 2.

Вообще говоря, в Вашем распоряжении две страницы видеопамяти. Вторая страница (номер 1) начинается на середине адресного пространства (а не сразу после страницы с номером 0), т.е. с адреса: (FFFFH DIV 2)+1=8000H. Вы можете писать непосредственно на эту страницу и, используя быстрое их переключение (см. Рис. 10.1), оживлять свои изображения.

III.

В вышеприведенных примерах точка ставится в заранее определенный байт. На практике, однако, задаются координаты точки и ее цвет. Следовательно, нужна еще процедура для расчета байта в видеобуфере и положение в байте (байт для регистра битовой маски). Смещение в буфере байта легко вычислить по формуле: (640*Y+X) DIV 8 или, упрощая выражение, получим 80*Y+X DIV 8. Пусть координата X находится в регистре CX, а координата Y - в регистре DX. Следующие ассемблерные команды решают данную проблему:

```
PUSH CX ;сохранить регистр CX
MOV AX,80       ;количество байт в строке
MUL DX          ;смещение с учетом целых строк
SHR CX,1        ;делим координату X на 8
SHR CX,1
SHR CX,1
ADD AX,CX       ;получаю смещение в буфере
POP CX          ;восстанавливаю регистр
```

В регистре AX будет находиться нужное смещение. Теперь найдем смещение в байте:

```
MOV  BX,AX
AND  CX,0111B ;получаю смещение в байте
MOV  AH,1
SHR AH,CL       /бит в месте, где стоит точка
```

Теперь в BX нужное смещение, а в AH байт для регистра битовой маски.

IV.

Рассмотрим теперь программу, демонстрирующую копирование страницы 0 на страницу 1. Процедура COPY сделана так, что ее можно использовать в других программах или в языках высокого уровня. Обращаю Ваше внимание, что режим записи 1 идеально подходит для целей копирования из одной области видеопамяти в другую. Как уже ранее отмечалось, страница 1 начинается со смещения 8000H видеобуфера.

```
CODE SEGMENT
       ASSUME CS:CODE
       ORG 100H
BEGIN:
;в графический режим
       MOV AX,0010H
       INT 10H
;вывод символа 1
       XOR BH,BH
       MOV DH,24
       MOV DL,20
       MOV AH,2
       INT 10H
       MOV BL,11        ;цвет символа
       MOV AL,65        ;код символа
       MOV AH,0AH
       MOV CX,1
       INT 10H
;копируем страницу 0 на страницу 1
       CALL COPY
;ждем нажатия клавиши
       MOV AH,0
       INT 16H
;вывод символа 2
       XOR BH,BH
       MOV DH,14
       MOV DL,20
       MOV AH,2
       INT ЮH
       MOV BL,09        ;цвет символа
       MOV AL,166       ;код символа
       MOV AH,0AH
       MOV CX,1
       INT 10H
;ждем нажатия клавиши
       MOV AH,0
```

```
        INT  16H
;показать  страницу  1
        MOV AH,05
        MOV AL,1
        INT ЮН
;ждем нажатия клавиши
        MOV  AH,0
        INT  16H
; в текстовый режим
        MOV  AX,0002H
        INT  10H
;выходим в DOS
        RET
;процедура копирования страницы 0 на страницу 1
COPY  PROC  NEAR
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH DX
        PUSH ES
;режим чтения-записи
        MOV   DX,3CEH
        MOV   AL,05
        OUT   DX,AL
        MOV  AL, 1           ;режим чтения 0, режим записи 1
        INC   DX
        OUT   DX,AL
        DEC   DX
;регистр сдвига
        MOV   AX,0003H
        OUT   DX,AX
;видеобуфер
        MOV   AX,0A000H
        MOV   ES,AX
        XOR   BX,BX
;количество байт
        MOV   CX,80*350
;регистр маски карты
        PUSH DX
        MOV   DX,3C4H
        MOV   AL,2
        OUT   DX,AL
        MOV   AL,0FH          ;разрешить все плоскости
        INC   DX
```

```
        OUT   DX,AL
POP   DX
;копируем
POVT:
        MOV   AL,ES:[BX]
        MOV   ES:[BX+8000H],AL
        INC   BX
        LOOP  POVT
;восстанавливаем режим чтения-записи
        MOV   AX,0005H
        OUT   DX,AX
;---  ---  ----  ----  --------
        POP   ES
        POP   DX
        POP   CX
        POP   BX
        POP   AX
        RET
COPY  ENDP
CODE  ENDS
        END  BEGIN
```

Рис. 10.10. Пример копирования видеостраницы 0 на видеостраницу 1.

Как видно из программы, мы используем в ней режим чтения 0 и режим записи 1. Это максимально подходит для всевозможных операций копирования.

V. Программа вывода на экран изображения в PCX формате.

Ниже представлена программа считывания изображения в PCX формате из файла на экран. Но прежде кратко изложим основу PCX-кодирования. Файлы в PCX формате имеют фиксированный заголовок длиной 128 байт. После заголовка идет само изображение в закодированном виде. Рассмотрим сначала заголовок. Его поля приводятся ниже. Для большей наглядности они представлены в ассемблерном формате.

```
        MANUF DB ? ;обычно 10
        HARD  DB ? ;номер версии, наше изложение касается вер-
сии 5
        ENCOD DB ? ;обычно 1, означает, что сжатие выполнялось
        BITPX DB ? ;число бит на точку (1 или 2)
   ;размеры картинки
        X1    DW ?
        Y1    DW ?
        X2    DW ?
```

```
    Y2     DW ?
    HRES   DW ? ;горизонтальное разрешение дисплея
    VRES   DW ? ;вертикальное разрешение дисплея
    PAL    DB 48 DUP(?) ;палитра
    VMODE DB ? ;резерв
    NPLAN DB ? ;количество плоскостей (обычно 4)
    BPLIN DW ? ;байт на строку
    PALINFO DW ? ;информация о палитре (1 - цв., 2-сер.)
;разрешение сканера, если изображение получалось с помощью
;этого устройства
    SHRES DW ?
    SVRES DW ?
    XTRA  DB 54 DUP(?) ;резерв
```

Если используется адаптер EGA или VGA в режиме эмуляции EGA, то палитра кодируется в 48 байтах (см. заголовок). Для специальных режимов VGA и SVGA палитра хранится после изображения. Суть кодирования палитры (см. расшифровку палитры в программе ниже) заключается в следующем. Как известно EGA-адаптер способен давать на экране одновременно 16 цветов. Каждый цвет составляется из трех чистых цветов: красного, зеленого и голубого. Интенсивность каждого из них определяется двумя битами. Таким образом, палитра для каждого цвета представляется шестибитным полем (см. выше) и соответственно может принимать 64 значения. В заголовке тройка байт представляет собой палитру для одного цвета (16*3=48). Каждый байт принимает значение от 0 до 255, причем значение от 0 до 63 дает нулевой уровень цвета, от 64 до 127 — первый уровень яркости и т.д.

Чтобы понять алгоритм сжатия изображения вспомним материал, изложенный выше. Последовательность из восьми точек кодируется четырьмя байтами. Каждый байт соответствует своей плоскости. Каждая строка изображения кодируется последовательностью байт: сначала для плоскости 0, затем 1 и т.д. Сжатие происходит для каждой строки пикселей. Если имеются повторяющиеся байты, то они кодируются следующим образом: вначале идет байт-повторитель, а затем повторяющийся байт. Признаком байта повторителя является наличие у него двух старших бит. Таким образом, можно закодировать до 64 повторяющихся байт. В случае неповторяющихся байт, поступают следующим образом: байт со значением, меньшим C0H, пишут в строку без изменения, в противном случае кодируют его с повторителем, равным 1.

Ниже представлена программа, производящая загрузку и раскодировку файла в PCX формате. Для того чтобы читателю проще было разобраться в программе, советую разделить ее на следующие части и разобраться с каждой отдельно: блок начальных установок, блок анализа командной строки, открытие файла и попытка выделения для него памяти, установка палитры, чтение файла в буфер, расшифровка с переносом в видеопамять. Особо обратите внимание на то, как обыгрывается вариант, когда длина файла больше 64К. В книге [12] указывается, что сжатая последовательность байт не может переходить границы строки (границу плоскости может). В других ис-

точниках, однако, такая возможность не исключается. Наш алгоритм построен таким образом, что будет правильно работать и в том, и в другом случае.

Для того чтобы разобраться с тем, как программа работает с памятью, Вам, возможно, придется обратиться к материалу главы 11.

Конечно, представленная здесь программа не учитывает все нюансы PCX формата[26] (см. [12]), но Вы можете усовершенствовать ее. В частности, следовало бы учесть возможность различных графических режимов, в том числе VGA и SVGA-режимов.

```
DATA  SEGMENT
PATH  DB, 80 DUP(0)
MES1  DB  'Ошибка памяти.',13,10,'$'
MES2  DB  'Ошибка при считывании.', 13,10,'$'
MES3 DB 'Ошибка в структуре файла.', 13,10, ' $ '
MES4  DB  'Нет имени файла.',13,10, '$'
MES5  DB  'Много параметров.',13,10,'$'
MES6  DB  'Файл не найден.',13,10,'$'
DATA  ENDS
SSEG  SEGMENT  STACK
       DB 100  DUP(?)
SSEG  ENDS
CODE  SEGMENT
      ASSUME  CS:CODE,DS:DATA,SS:SSEG
BEGIN:
;начальные установки
;обрезать выделенную программу по концу программы
      MOV AX,DS
      MOV BX,SEG RR
      SUB BX,AX
      MOV AH,4AH
      INT 21H
;установка сегментных регистров
      MOV AX,DATA
      MOV DS,AX
      MOV AX,SSEG
      MOV SS,AX
;анализ командной строки
      MOV DL,1
      XOR SI,SI
      XOR DI,DI
```

[26] **В принципе PCX** формат **уже** устарел, **и им пользуются достаточно редко. Я вижу две основные причины: 1. Изначально в формате не были заложены возможные** пути **развития графических адаптеров. 2. Механизм сжатия, используемый в PCX формате, достаточно** слаб и **значительно уступает сжатию в таких форматах, как GIF, JPG и др.**

```
PROO:
       CMP BYTE PTR ES: [81H] [SI],0DH
       JZ  KON_1
       CMP BYTE PTR ES:[81H][SI],32
       JZ  KON_6
       CMP DL, 2
       JNZ KON_4
       MOV DL,1
       JMP SHORT KON_2
KON_4:
       MOV AL,ES:[81H][SI]
       MOV PATH[DI],AL
       INC DI
       MOV DL,0
KON_5:
       INC SI
       JMP SHORT PROO
KON_6:
       CMP DI,0
       JZ  KON_5
       MOV DL,2
       JMP SHORT KON_5
KON_1:
       CMP DI,0
       JNZ KON_2
       LEA DX,MES4
       JMP PROD3
KON_2:
       CMP DL,1
       JNZ KON_3
       LEA DX,MES5
       JMP PROD3
KON_3:
;установим графический режим экрана
       MOV AH,0
       MOV AL,10H
       INT ЮН
;вызов процедуры загрузки и вывода картины
       LEA DX,DS:PATH
       CALL LOAD_PCX
       PUSH AX
       CMP AH,0
       JNZ ERR
```

```
;ждем нажатия клавиши
        MOV AH,0
        INT 16H
;возвратимся в текстовый режим
ERR:
        MOV AH,0
        MOV AL,3
        INT ЮН
        POP AX
        CMP AH,0
        JZ  WYH
        CMP AH,1
        JNZ PRODI
        LEA DX,MES1
        JMP SHORT PROD3
PROD1:
        CMP AH,2
        JNZ PROD2
        LEA DX,MES2
        JMP SHORT PROD3
PROD2:
        LEA DX,MES3
PROD3:
;вывод строки
        MOV AH,9
        INT 21H
;выход в ДОС
WYH:
        MOV AH,4CH
        INT 21H
;процедура считывания и расшифровки PCX файла
;вход DS:DX - путь к файлу, в BL 0 или 1 (страницы)
;выход AH - 0 -нормально, 1-ошибка памяти, 2-ошибка считывания,
;3-ошибка структуры
LOAD_PCX PROC
        PUSH DS
        PUSH ES
        PUSH BX
        PUSH CX
        PUSH DX
;открыть файл
        MOV AX,3D00H
        INT 21H
        JNC NORM1
```

```
        MOV AH,2
        JMP KON
NORM1:
        MOV BX,AX ;описатель в BX найдем длину
        MOV AX,4202H
        XOR CX,CX
        XOR DX,DX
        INT 21H          ;длина в DX:AX
        MOV CX,16
        DIV CX
        INC AX
;теперь в AX количество необходимых параграфов
        MOV CS:_PAR,AX
;теперь на начало файла
        MOV AX,4200H
        XOR CX,CX
        XOR DX,DX
        INT 21H
;читаем первые 128 байт (заголовок)
        MOV CX,128
        MOV AH,3FH
        PUSH CS
        POP  DS
        LEA  DX,CS:BLOK
        INT  21H
        CMP  CX,AX
        JZ   NORM2
        MOV  AH,2
        JMP  KON
NORM2:
;теперь попробуем выделить буфер
        PUSH BX
        MOV  BX,CS:_PAR
        MOV  AH,48H
        INT  21H
        POP  BX
        JNC  NORM3
;буфер выделить не удалось
        MOV  AH,1
        JMP  KON
NORM3:
        MOV CS:_SEG,AX
;установка палитры
        PUSH BX
```

```
        XOR   BL,BL
        LEA  SI,CS:PAL
MOV CX,16
PAL_D:
        PUSH CX
;RED
        XOR   AH,AH
        MOV   AL,CS:[SI]
        MOV   CL,6
        SHR   AL,CL
        MOV   CL,2
        DIV   CL
        MOV   CL,5
        SHL   AH,CL
        MOV   CL,2
        SHL   AL,CL
        ADD   AL,AH
        MOV   BH,AL
;GREEN
        XOR   AH,AH
        MOV   AL,CS:[SI+1]
        MOV   CL,6
        SHR   AL,CL
        MOV   CL,2
        DIV   CL
        MOV   CL,4
        SHL   AH,CL
        SHL   AL,1
        ADD   AL,AH
        ADD   BH,AL
; BLUE
        XOR   AH,AH
        MOV   AL,CS:[SI+2]
        MOV   CL,6
        SHR   AL,CL
        MOV   CL,2
        DIV   CL
        MOV   CL,3
        SHL   AH,CL
        ADD   AL,AH
        ADD   BH,AL
;вызов функции установки палитры
        MOV   AX,1000H
        INT   ЮН
```

```
        INC  BL
        ADD  SI,3
        POP  CX
        LOOP PAL_D
        POP  BX
;---     --------
;теперь в DS сегментный адрес буфера
        MOV DS,CS:_SEG
        XOR DX,DX
POVT:
        MOV CX,60000
        MOV AH,3FH
        INT 21H
        CMP AX,CX
        JNZ NORM4
;делим на 16, чтобы определить добавку к DS
        MOV CX,DS
        ADD CX,3750   ;параграфы, а в DX остаток
;(вообще говоря 0)
        MOV DS,CX
        mov AL,0
        JMP SHORT POVT
NORM4:
        MOV AX,CS:_SEG
        MOV DS,AX
;закроем файл
        MOV AH,3EH
        INT 21H
;здесь копируем в видеобуфер
;теперь файл в буфере количество строк
        MOV AX,CS:Y2
        SUB AX,CS:Y1
        INC AX
        MOV CS:KOL_STROK,AX
;здесь обработка, если не PAINTBRUSH
NORM5:
        MOV AX,CS:X2
        SUB AX,CS:X1
        INC AX
        MOV CL,8
        DIV CL
        MOV CS:KOL_BYTE,AL
;здесь определяем смещение в видеобуфере
        MOV AX,CS:X1
        MOV CL,8
```

```
        DIV  CL
        XOR  AH,AH
        PUSH AX
        XOR  DX,DX
        MOV  CX,80
        MOV  AX,CS:Y1
        MUL  CX
        POP  BX
        ADD  BX,AX
        MOV  CS:_BX,BX
;----------------
        MOV  CL,CS:NPLAN
        SHL  CS:N_PL,CL
;готовим  регистры
        MOV  DX,3CEH
        MOV  AX,0005H
        OUT  DX,AX
        MOV  AX,0003H
        OUT  DX,AX
        MOV  AX,0FF08H
        OUT  DX,AX
        MOV  AX,0A000H
        MOV  ES,AX
        XOR  SI,SI
;здесь начинается раскрутка, при этом ES:BX на видеобуфер,
;DS:SI на содержимое файла, KOL_STROK - количество строк
        MOV  AL,2
        MOV  CX,CS:KOL_STROK
LOOP1:          ;цикл строк
        PUSH CX
        CMP  SI,60000
        JB   KO
        PUSH AX
        PUSH DX
        PUSH CX
        PUSH BX
        MOV  AX,SI
        XOR  DX,DX
        MOV  CX,16
        DIV  CX
        MOV  BX,DS
        ADD  AX,BX
        MOV  DS,AX
        MOV  SI,DX
```

```
        POP   BX
        POP   CX
        POP   DX
        POP   AX
KO:
        MOV   AH,1
LOOP2:          ;цикл плоскостей
        PUSH  AX
        MOV   DI,CS:BPLIN ;количество байт для
;одной плоскости
        MOV DX,3C4H
        OUT   DX,AX             ;устанавливаем плоскость
        MOV   DL, CS:KOL_BYTE
        MOV   BX,CS:_BX
LOOP3:                                  ;цикл в одной плоскости
;----------------
        MOV CL,DS:[SI]
        CMP CL,OCOH
        JB   LOOP4
;с повторителем
        SUB CL,0C0H
        MOV CH,DS:[SI+1]
LOOP5:
        CMP   DL,0
        JZ    LOY
        MOV   ES:[BX],CH
        INC   BX
        DEC   DL
LOY:
        DEC   DI
        JZ    WOWO
        DEC   CL
        JNZ   LOOP5
        ADD   SI,2
        JMP   SHORT LOOP3
WOWO:
        DEC   CL
        JNZ   WOWOWO
        ADD   SI,2
        JMP   SHORT LOOP6
WOWOWO:
        ADD   CL,0C0H
        MOV   DS:[SI],CL
        JMP   SHORT LOOP6
```

```
LOOP4:                          ;один байт
        CMP   DL,0
        JZ    LOOP7
        MOV   ES:[BX],CL
        INC   BX
        DEC   DL
LOOP7:
        INC   SI
        DEC   DI
        JNZ   LOOP3
;----------------
LOOP6:
        POP   AX
        SHL   AH,1
        CMP   AH,CS:N_PL
        JNZ   LOOP2
        POP   CX
        DEC   CX
        JZ    KONN
        MOV   BX,CS:_BX
        ADD   BX,80
        MOV   CS:_BX,BX
        JMP   LOOP1
KONN:
        MOV BX,DS
        MOV ES,BX
        MOV AH,49H
        INT  21H
        MOV   AH,0
        JMP   SHORT KON
; во временный буфер
;закроем буфер
CLOS_BUF:
        MOV BX,DS
        MOV ES,BX
        MOV AH,49H
        INT  21H
KON:
        POP  DX
        POP  CX
        POP  BX
        POP  ES
        POP  DS
        RET
```

```
KOL_STROK DW ?              ;количество строк
_BX       DW ?              ;начало буфера (в байтах) =
;=(X1/8)+Y1*80
KOL_BYTE  DB ?
N_PL      DB 1
_PAR      DW ?        ;количество необходимых параграфов
_SEG      DW ?        ;сегментный адрес буфера
;заголовок PCX файла
BLOK:
MANUF     DB ?       ;10 для PAINTBRUSH
HARD      DB ?       ;информация о версии (5)
ENCOD     DB ?       ;закодировано (1)
BITPX     DB ?       ;бит на точку (4)
;координаты картинки
X1        DW ?
Y1        DW ?
X2        DW ?
Y2        DW ?
HRES      DW ?       ;горизонтальное разрешение
VRES      DW ?       ;вертикальное разрешение
PAL       DB 48 DUP(?)  ;палитра
VMODE     DB ?       ;игнорируется
NPLAN     DB ?       ;количество плоскостей
BPLIN     DW ?       ;байт на строку
PALINFO   DW ?       ;информация о палитре
SHRES     DW ?       ;разрешение сканера
SVRES     DW ?
XTRA      DB 54 DUP(?)
LOAD_PCX ENDP
CODE ENDS
;сегмент для определения конца программы
ZSEG SEGMENT
RR   DB ?
ZSEG ENDS
     END BEGIN
```

Рис. 10.11. Пример вывода па экран файла в PCX формате (16 цветов).

Глава 11. Работа с памятью.

Память человека есть лист белой бумаги. Иногда напишешь хорошо, а иногда дурно.

Козьма Прутков.

Программа (программный код, данные и стек) после загрузки в память занимает лишь часть (иногда незначительную) ее свободного пространства. Естественно, что в нашей власти использовать доступную память по своему усмотрению. О том, как сделать это корректно, будет рассказано в данной главе. Сразу оговорюсь, что речь идет о так называемой обычной памяти (английский термин conventional), мы назвали ее базовой (см. главу 2). О расширенной (extended), дополнительной (expanded) и других видах памяти в операционной системе MS DOS речь пойдет в других главах (см. главу 22, а также главы 5 и 20).

I.

Основу для манипуляции памятью составляют три функции 21Н-го прерывания: 48H, 49H, 4AH. В главе 1 мы говорили о блоковой структуре памяти, с которой работает DOS. Данные функции как раз и манипулируют указанными блоками памяти. Ниже дается подробное описание данных функций. Конкретный пример использования функций 48Н и 49Н Вы сможете найти в главе 12 (см. также конец главы 10). Блоками памяти можно манипулировать, и минуя указанные функции (вручную). Это достаточно сложная процедура, в главе 12 Вы сможете познакомиться с примерами такой манипуляции.

Итак вот эти функции:

48Н - выделяет блок памяти. Размер запрашиваемой памяти должен быть в ВХ. Если блок такого размера существует, то в АХ возвращается сегментный адрес выделенного блока. Если флаг С установлен, то в АХ помещается код ошибки, а в ВХ — максимально доступный объем памяти. Размеры блоков памяти измеряются в параграфах.

49Н - освобождает блок памяти, который становится доступным для других процессов. В ES должен находиться сегментный адрес освобождаемого блока. Если флаг С установлен, то в АХ содержится код ошибки.

4АН - сжимает или расширяет выделенный блок. В ES — сегментный адрес блока, в ВХ — желаемый размер. Если флаг С установлен, то в АХ — код ошибки, а в ВХ — доступный наибольший блок (при попытке расширения).

Когда запускается программа, то ей автоматически распределяется весь доступный объем памяти. Однако если Вы пожелаете запустить какую-то другую программу с помощью стандартных DOS'овских процедур, Вам придется освободить часть памяти.

II.

Оставим пока в стороне запуск программы и поговорим подробнее о том, как можно использовать свободную память. Первый вопрос, который возникает в этой связи, - как

добраться до свободной памяти, находящейся в старших адресах? Ответ на этот вопрос состоит из двух частей:

1. Нужно знать параграф, откуда эта память начинается.
2. Для того чтобы знать параграф, необходимо, чтобы среди сегментов программы был один пустой, начинающийся с буквы, которая идет в алфавите после всех первых букв других сегментов. Например:

```
ZSEG   SEGMENT
ZSEG   ENDS
```

Если такой сегмент есть, то необходимо транслировать программу с ключом /a (или использовать директиву .ALPHA). В этом случае все сегменты будут выстроены по алфавиту, и пустой сегмент окажется последним. По умолчанию MASM располагает сегменты в том порядке, как они идут в тексте программы, поэтому можно поместить ZSEG в конце программы и транслировать ее без всяких опций. Выполнив две команды: MOV AX,ZSEG и MOV ES,AX, мы получим в ES как раз параграф, откуда начинается свободная память. Теперь ее можно свободно использовать. Помните только, что память ограничена, и если Вы намерены написать хорошую программу, которая бы контролировала свои возможности и резервы, то лучше пойти другим путем. Он состоит в следующем. Нужно урезать размер отведенной области до размеров самой программы. После этого Вы можете запрашивать необходимую память с помощью 48Н-Й функции. В случае, если затребованная память слишком велика, то функция возвратит в ВХ свободное количество памяти в параграфах. Таким образом, Вы всегда сможете контролировать ситуацию. Рассмотрим, как происходит урезание памяти.

```
; при запуске программы ES и DS  указывают на начало PSP
; будем предполагать, что значение ES  не менялось[27]
      MOV   AX,ZSEG
      MOV   BX,ES
      XCHG  AX,BX
      SUB   BX,AX
      MOV   AH,4AH
      INT   21H
```

Не забывайте, что транслировать программу надо с ключом /a. Сегмент может располагаться как в начале, так и в конце программы. Проблема может возникнуть в случае компоновки нескольких модулей, но разговор об этом мы отложим до главы 13.

До сих пор мы говорили об ЕХЕ-программах. Рассмотрим, как обстоят дела в случае СОМ-программ. Вся СОМ-программа помещается в один сегмент. При запуске программы все четыре сегментных регистра будут указывать на начало этого сегмен-

[27] Функция DOS 62Н возвращает в ВХ сегментный адрес PSP выполняющейся программы.

та. На конец сегмента будет указывать регистр стека – SP, стек будет расти в сторону программы. Все адресное пространство за текущим сегментом будет в Вашем распоряжении. В главе 4 (см. Рис. 4.6) приведен пример использования этого пространства. Можно использовать пространство и непосредственно за программой. Для этого достаточно в конце программы поставить метку и обращаться к адресам относительно этой метки. Только надо иметь в виду, что при этом Вы попадаете непосредственно в область стека со всеми вытекающими отсюда последствиями. Впрочем, стек можно расположить и в другом месте.

III.

Перейдем к рассмотрению вопроса запуска одной программы из другой. Данная проблема решается с использованием функции DOS 4BH. Эта функция позволяет не только запускать EXE или COM-программы, но и загружать оверлей. Что касается оверлеев, то здесь можно довольно легко обойтись без указанной функции, что мы сейчас и сделаем.

Рассмотрим сначала программу на Рис. 11.1, которая и являет собой простейший пример оверлейного модуля. По сути дела, это обычная COM-программа с той лишь разницей, что здесь стоит ORG 0, а не ORG 1(Ю)Н. Но это должно быть понятно, ведь у оверлея нет PSP. После работы редактора связей LINK.EXE необходимо преобразовать EXE-модуль к COM-формату (не обязательно с расширением COM). Второй момент, на который я хотел бы обратить Ваше внимание, это работа с сегментными регистрами. После передачи управления оверлею на его сегмент будет указывать CS. Остальные сегменты будут иметь значения такие, какие имели в основной программе. В нашем примере мы загружаем оверлей в сегмент данных, на который указывает DS (хотя могли бы загрузить и в адресное пространство за программой). Воспользовавшись этим, мы не переопределяем DS. Правильнее было бы поместить в DS значение из CS, а в конце, естественно, восстановить DS.

```
CODE  SEGMENT
        ASSUME  CS:CODE
        ORG 0
BEGIN:
        LEA  DX,CS:TEXT
        MOV AH,9
        INT  21H
        RETF
TEXT  DB 'Привет',13,10,'$'
CODE  ENDS
      END
```

Рис. 11.1. Оверлей к программе на Рис. 11.2.

Рассмотрим теперь Рис. 11.2. Обращаю Ваше внимание на то, что оверлей мы считываем как обычный файл. Как можно заметить, для создания оверлейной струк-

туры, вообще говоря, не обязательно пользоваться специальными средствами (функция 4BH). Используя такой метод, можно создать простейший менеджер управления оверлеями[28].

```
DSEG  SEGMENT
      DB 18 DUP(0)                ;буфер для оверлея
PATH DB 'PR1.COM',0               ;имя оверлея
DSEG  ENDS
SSEG  SEGMENT  STACK
      DB 30 DUP(?)
SSEG ENDS
CODE  SEGMENT
      ASSUME CS:CODE, DS:DSEG, SS:SSEG
BEGIN:
      MOV AX,DSEG
      MOV DS,AX
      MOV AH,3DH
      MOV AL,0
      LEA DX,PATH
      INT 21H                     ;открываем оверлей для чтения
      JC   EXIT
      MOV BX,AX
      XOR DX,DX
      MOV AH,3FH
      MOV CX,18
      INT 21H                     ;читаем оверлей в буфер
      MOV AH,3EH
      INT 21H
      CALL DWORD PTR  CS:[OVERL]   ;осуществляем запуск
      ;процедуры в оверлее
EXIT:  ;выход из программы
      MOV AH,4CH
      INT 21H
```

[28] Менеджер управления оверлеями в современных языках высокого уровня - довольно сложная процедура. В ее задачу входит обслуживание кольцевого оверлейного буфера. Проблема заключается в том, что при вызове процедуры из оверлея менеджер должен определить, не находится ли данный оверлей уже в памяти - если находится, то с диска читать не надо. Если в памяти его нет, то следует прочесть его в буфер. Но тут возникает вторая проблема: в буфере может не быть места. Следовательно, какой-то другой оверлей должен быть вытолкнут из очереди. Часто таким оказывается первый из очереди. Однако современные оверлейные менеджеры «умудряются» еще следить за частотой вызова того или иного оверлея, и вытолкнутым оказывается редко вызываемый оверлей. Добавьте сюда еще постоянную настройку адресов, по которым вызываются оверлейные процедуры. Как видим, все не так просто.

```
;полный адрес/ где будет располагаться оверлей
OVERL DW 0,                             ;смещение
      DW SEG PATH                       ;сегмент
CODE ENDS
      END BEGIN
```

*Рис. 11.2. Пример запуска оверлея (см. Рис. 11.1) без использования функции **4BH**.*

IV.

Рассмотрим работу функции 4BH. В силу важности и сложности данной функции здесь дается ее полное описание.

AH 4BH
DS:DX адрес строки с именем программы, строка может содержать полный путь к каталогу, где находится программа, в конце строки должен стоять 0.
ES:BX адрес блока параметров (см. ниже)
AL 0-загрузить и выполнить программу,
3-загрузить оверлей.

Если после выполнения данной функции флаг С будет восстановлен, то в регистре AX будет содержаться код ошибки. По поводу загрузки оверлеев может сразу возникнуть вопрос: зачем нужна эта функция, если оверлей можно загрузить как обычный файл (что мы только что сделали)? Ответ очень прост: оверлей может иметь структуру EXE-программы. Разбираться в заголовке и самому настраивать адреса - Боже упаси! Рассмотрим структуру блока параметров. Вначале разберем блок параметров для запуска программы (Рис. 11.3).

Смещение	Длина	Содержимое
O	2	Сегмент окружения. Если 0, то задача наследует окружение родителя. Можно придумать свое собственное окружение.
2	4	Адрес командной строки для запускаемой программы. Данная строка затем помещается по адресу 80H в PSP. В начале строки должен стоять байт длина. Строка должна заканчиваться кодом 13.
6	4	Адрес блока FCB для помещения в PSP по адресу 5CH.
10	4	Адрес блока FCB для помещения в PSP по адресу 6CH.
14	1	Длина блока параметров.

Pud. 11.3. Блок параметров для запуска программы.

FCB в 6-13 байтах блока параметров должны содержать информацию о файлах, указанных в командной строке. Если таковых нет, или если программа не работает с FCB, то указанные байты заполняются нулями.

Ниже (Рис. 11.4) представлен блок параметров для загрузки оверлея.

Смещение	Длина	Содержимое
0	2	Сегмент, в который будет загружен файл.
2	4	Фактор перемещения, для корректировки сегментных ссылок.
6	1	Длина блока параметров.

Рис. 11.4. Блок параметров для загрузки оверлея.

Вы можете загрузить оверлей как во внутренний сегмент, так и вне программы. Если Вы загружаете оверлей вне программы, то сначала освободите для этого место. Фактор привязки дает смещение для настройки адресов (например, в командах MOV BX,SEG MEM, длинный переход и др.). Помещайте в фактор привязки сегментный адрес оверлея.

```
DATA   SEGMENT
F_NAME   DB 'C:\DOS\FORMAT.COM',0
PARAM DW   0
      DW   OFFSET STR      ;указываем на строку параметров
      DW   SEG STR
FC1   DW   OFFSET FCB1     ;указываем на первый FCB
      DW   SEG FCB1
FC2   DW   OFFSET FCB2     ;указываем на второй FCB
      DW   SEG FCB2
FCB1  DB   40 DUP(0)
FCB2  DB   40 DUP(0)
STR   DB   2,'A:',13        ; строка параметров для программы
_SS   DW   ?
_SP   DW   ?
TEXT  DB  'Программа FORMAT.COM закончила свою работу.', 13,10,'$'
DATA       ENDS
;---------------------
SSEG       SEGMENT STACK
      DB 50 DUP(?)
SSEG       ENDS
;---------------------
CODE       SEGMENT
      ASSUME CS:CODE, DS:DATA, SS:SSEG, ES:DATA
BEGIN:
      MOV AX,DATA
      MOV DS,AX
```

```
        MOV BX,SEG  ZSEG
        MOV AX,ES          ;ES указывает на начало PSP
        SUB BX,AX
        MOV AH,4AH
        INT 21H            ;обрезаем программу
;--указываем на блок параметров
        MOV AX,SEG PARAM
        MOV ES,AX
        LEA BX,PARAM
;--сохранить копии стековых регистров
        MOV _SS,SS
        MOV _SP,SP
;--указываем на строку имени
        LEA  DX,F_NAME
;--загрузка и выполнение программы
        MOV AX,4B00H
        INT 21H
;--восстанавливаем сегментные регистры
        MOV AX,DATA
        MOV DS,AX
        MOV SS,_SS
        MOV  SP,_SP
        JC   EXIT
;--печатаем строку,  если все прошло успешно
        LEA DX,TEXT
        MOV AH,9
        INT 21H
EXIT:
        MOV  AH,4CH
        INT  21H
CODE       ENDS
ZSEG       SEGMENT
ZSEG       ENDS
        END BEGIN
```

Рис. 11.5. Программный запуск утилиты FORMAT.COM.

На Рис. 11.5 представлен пример запуска программы из другой программы. В качестве запускаемой программы выбрана известная утилита DOS FORMAT.COM. Обращаю Ваше внимание на то, что в данном случае нам понадобилось резервировать место для блоков FCB [29]. Кроме того, необходимо указать строку параметров для данной программы, так как она помещается в PSP с адреса 80H (в начале — длина, затем сами

[29] Использование FCB для работы с файлами - устаревший метод. Однако некоторые утилиты DOS пользуются этим методом (см.[5]).

параметры, в конце — 0DH). Если бы программа не использовала FCB и не требовала бы параметров, то можно было ограничиться всего одной строкой: PARAM DB 14 DUP(0).

V.

Перейдем к рассмотрению использования функции DOS 4BH для загрузки оверлея. На Рис. 11.6 приведена программа, которая будет играть роль оверлея. В принципе, это обычная EXE-программа. Есть, однако, небольшой нюанс: выход из нее осуществляется по RETF. Если запустить программу из операционной системы, то, выполнив предназначенные для нее действия, она не сможет выйти обратно - "зависнет". Второй момент, на который я хочу обратить Ваше внимание, это то, что в оверлее определен стек. Однако при вызове оверлея регистры SS и SP указывают на стек основной программы, в оверлее же мы не меняем их значения. Поэтому в принципе свой стек в оверлее не нужен. А вот если бы мы не направили регистр DS на сегмент данных, то он бы показывал на сегмент данных основной программы, и желательного эффекта мы бы не добились. Наконец, последнее, что я должен Вам сообщить по программам на Рис. 11.6-11.7. Обе программы необходимо транслировать с опцией /а. Для основной программы это необходимо, чтобы сегмент ZSEG стал последним, а для оверлея такая необходимость связана с тем, что в этом случая первым будет идти сегмент кода, и адрес OVR_SEG:OVR_OFF в основной программе будет как раз указывать на начало программы, т.е. BEGIN[30]. Попробуйте оттранслировать оверлей без указанной опции, поставив сегмент данных впереди, и оверлей перестанет запускаться.

```
DATA  SEGMENT
TEXT  DB  'Привет!',13,10,'$'
DATA  ENDS
SSEG  SEGMENT  STACK
       DB 50 DUP(?)
SSEG  ENDS
CODE  SEGMENT
       ASSUME CS:CODE, DS:DATA, SS:SSEG
BEGIN:
       MOV AX,DATA
       MOV DS,AX
       LEA DX,TEXT
       MOV AH, 9
       INT 21H
       RETF                ;возврат из оверлея
CODE  ENDS
       END BEGIN
```

Рис. 11.6. Программа-оверлей, загружаемая из программы на Рис. 11.7.

[30] Это будет выполняться, если тип подгонки сегмента кода PARA (см. главу 13), что устанавливается по умолчанию.

```
DATA  SEGMENT
PARAM    DB 4 DUP(0)
PATH DB ' PR1.EXE',0
OVR_OFF     DW 0
OVR_SEG   DW ?
STR_ER    DB 'Ошибка при загрузке оверлея.',13,10, '$'
STR        DB 'Оверлей закончил свою работу.',13,10,'$'
DATA   ENDS
SSEG        SEGMENT STACK
      DB 50 DUP(?)
SSEG        ENDS
;----------------------
CODE        SEGMENT
      ASSUME CS:CODE, DS:DATA, SS:SSEG, ES:DATA
BEGIN:
      MOV AX,DATA
      MOV DS,AX
      MOV BX,SEG ZSEG
      MOV AX,ES            ;ES указывает на начало PSP
      SUB BX,AX
      MOV AH,4AH
      INT 21H              ;обрезаем программу
;выделяем для программы 2000 байт
      MOV BX,2000
      MOV AH,48H
      INT 21H
      JC   ERR             ;если не хватает памяти, то выход
      MOV OVR_SEG,AX
;готовим регистры для загрузки
      MOV   AX,DATA
      MOV   ES,AX
      LEA   BX,PARAM
      MOV   AX,OVR_SEG
      MOV    [BX],AX      ;сегмент, куда загружается оверлей
      MOV    [BX]+2,AX   ;фактор привязки совпадает с сегментом
      LEA   DX,PATH
      MOV   AL,3
      MOV   AH,4BH
      INT   21H
      JC    ERR
      CALL  DWORD PTR DS:OVR_OFF   ;вызов оверлея
;----------------------------
      MOV   AX,DATA
      MOV   DS,AX
```

```
        LEA   DX,STR
        MOV   AH,9
        INT   21H
EXIT:
        MOV   AH,4CH
        INT   21H
ERR:
        LEA  DX,STR_ER
        MOV  AH,9
        INT  21H
        JMP  EXIT
CODE         ENDS
ZSEG         SEGMENT
ZSEG         ENDS
        END  BEGIN
```

Рис. 11.7. Программа, загружающая оверлей (см. Рис. 11.6).

До сих пор мы говорили о "легальной" работе с памятью - посредством функций DOS. В главе 2 было дано понятие блоков управления памятью. Это недокументированные сведения. Однако, судя по всем признакам, такая структура памяти останется в силе и в дальнейшем. Управлять памятью (занимать память, освобождать память и т.д.) можно, непосредственно работая с этими блоками. И хотя это гораздо сложнее, есть ситуации, когда такой подход необходим. В следующих главах будут даны конкретные примеры (см. также раздел VII данной главы).

VI.

Наконец пришло время поговорить о завершении программы. Функция 4CH является универсальной. Ею можно завершать как COM, так и EXE-программы. Кроме того, она позволяет возвращать код выхода родительскому процессу. Однако есть более старые и традиционные способы завершения программ.

В первых двух байтах PSP стоит команда INT 20H. Это традиционная команда завершения программы. Однако, для того чтобы эта команда сработала, необходимо, чтобы регистр CS указывал на начало PSP. Для COM-программы это так, и завершать ее можно, выдав команду INT 20H. Для EXE-программы это обычно не так, и, чтобы выйти из положения, можно сделать длинный JMP (JMP FAR) на начало PSP. При запуске EXE-программы DS и ES указывают на PSP, поэтому проблем в нахождении этого сегмента не существует. Подготовьте в сегменте данных четыре байта:

```
PSP_OFF DW 0        ;смещение 0 в сегменте PSP
PSP_SEG DW ?
```

В начале программы выполните команду MOV PSP_SEG,ES (DS указывает уже на сегмент данных). В конце же программы выполните команду:

```
        JMP DWORD PTR DS:PSP_OFF
```

и программа будет завершена. Замечу, что после запуска программы в стек помещается слово 0. Это означает, что COM-программу можно завершить просто командой RET (!).

Наконец, еще есть функция DOS 0, которая фактически идентична прерыванию 20H. На адрес возврата из программы направлен также вектор 22H.

О том, как завершить программу и оставить ее при этом в памяти (резидентно), мы поговорим в следующей главе.

VII.

В главе 2 мы кратко коснулись структуры управляющего блока памяти MCB. В данной главе приводится полезная и простая программа, с помощью которой можно провести исследование основной памяти на предмет управляющих блоков.

```
DATA  SEGMENT
STR0      DB 'Конец памяти' ,13,10,'$'
STR1      DB 'Сегмент блока', 13,10,'$'
STR2      DB 'Длина блока в параграфах' ,13,10,'$'
STR3      DB 'Свободный',13,10,'$'
STR4      DB 'Параграф владельца' ,13,10,'$'
STR5      DB '-------------------------',13,10,'$'
STR6      DB 'Имя владельца' ,13,10,'$'
STR7      DB 'Адрес последнего параграфа' ,13,10,'$'
;строка для вывода шестнадцатеричного числа
STROKA    DB '0000',13,10,'$'
;шаблон
SHABL     DB  '0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'
DATA ENDS
;сегмент стека
STECK  SEGMENT  STACK
       DB  100  DUP(?)
STECK  ENDS
;сегмент кода
CODE  SEGMENT
       ASSUME CS:CODE, DS:DATA, SS:STECK
BEG:
       MOV   AX,DATA
       MOV   DS,AX
       CALL  BEG_MEM
PROD:
;сегментный адрес
       LEA   DX,STR1
       CALL  TEXT
       MOV   AX,ES
```

```
        INC   AX
        CALL HEX
;длина блока в параграфах
        LEA   DX,STR2
        CALL TEXT
        MOV   AX,ES:[3]
        CALL HEX
;параграф владельца
        LEA   DX,STR4
        CALL TEXT
        MOV   AX,ES:[1]
        CALL HEX
;имя владельца
        LEA   DX,STR6
        CALL TEXT
        CALL NAM
        LEA   DX,STR5
        CALL TEXT
;следующий блок?
        CALL NEXT
        CMP  AL, 0
        JZ    PROD
        LEA   DX,STR0
        CALL TEXT
        LEA   DX, STR7
        CALL TEXT
        MOV   AX,ES
        ADD   AX,ES:[3]
        MOV   ES,AX
        CALL HEX
KONEC:
        MOV AH,4CH
        INT 21H
;раздел процедур
;адрес первого блока в ES
BEG_MEM  PROC
        MOV AH,52H
        INT 21H
        MOV ES,ES:[BX-2]
        RET
BEG_MEM ENDP
;вывод шестнадцатеричного числа, находящегося в AX
HEX      PROC
        XOR DX,DX
        MOV BX,1000H
```

```
        DIV  BX
        MOV  BX,AX
        MOV  BL,BYTE PTR SHABL[BX]
        MOV  STROKA,BL
        MOV  AX,DX
        XOR  DX,DX
        MOV  BX,100H
        DIV  BX
        MOV  BX,AX
        MOV  BL,BYTE PTR SHABL[BX]
        MOV  STROKA+1,BL
        MOV  AX,DX
        XOR  DX,DX
        MOV  BX,10H
        DIV  BX
        MOV  BX,AX
        MOV  BL,BYTE PTR SHABL[BX]
        MOV  STROKA+2,BL
        MOV  BX,DX
        MOV  BL,BYTE PTR SHABL[BX]
        MOV  STROKA+3,BL
        LEA  DX,STROKA
        CALL TEXT
        RET
HEX     ENDP
;процедура перехода к следующему блоку
;адрес в ES, если в AL 0, если 1, тогда блоков больше нет
;если 2, то текущий блок памяти разрушен
NEXT    PROC
        MOV  AL,1
        CMP  BYTE PTR ES:[0],'Z'
        JZ   NET
        CMP  BYTE PTR ES:[0],'M'
        JZ   NET1
        MOV  AL,2
        JMP SHORT NET
NET1:
        MOV  AX,ES
        INC  AX
        ADD  AX,ES:[3]
        MOV  ES,AX
        XOR  AL,AL
NET:
        RET
NEXT    ENDP
```

```
;имя блока
NAM     PROC
        CMP WORD PTR ES: [1] , 0
        JZ  WWWW              ;не свободный ли блок?
        PUSH ES
        MOV DI,ES
        MOV SI,ES:[1]
        DEC SI
        CMP SI,DI
        JZ  WO
        MOV ES,SI
WO:
        XOR BX,BX
        MOV AH, 2
WW:
        MOV DL,ES:[BX][8]
        CMP DL,65
        JB  WWW
        INT 21H
        INC BX
        JMP SHORT WW
WWW:
        MOV DL,13
        INT 21H
        MOV DL,10
        INT 21H
        POP ES
        RET
WWWW:
        LEA DX,STR3
        CALL TEXT
        RET
NAM     ENDP
;вывод строки
TEXT PROC
        MOV AH, 9
        INT 21H
        RET
TEXT ENDP
CODE     ENDS
        END BEG
```

Рис. 11.8. Программа исследования блоковой структуры основной памяти.

Данную программу - назовем ее MEMO - удобно использовать, перенаправляя вывод в файл. Например, MEMO > MEMO.LST. Обращаю Ваше внимание на то, как в

программе находится адрес заголовка первого управляющего блока. Функция 52H возвращает указатель на блок DOS'овских переменных, который помещается в паре регистров ES:BX. В частности, там содержится и необходимый нам адрес. Обращаю также внимание и на то, что процедура NEXT может возвращать в AL и 2, что означает ошибку в структуре блока.

VIII.

Зададимся теперь одним интересным вопросом: можно ли использовать PSP в своих целях, например, для хранения данных или кода программы? Это может иметь смысл, например, для резидентных программ (см. главу 12). Вообще говоря, портить PSP нежелательно: там хранятся важные данные, в частности, необходимые для правильного завершения программы. Сейчас будет показано, как можно справиться с этими трудностями.

```
CODE   SEGMENT
       ASSUME CS:CODE, DS:CODE, SS:CODE, ES:CODE
       ORG 100H
BEGIN:
;найти сегментный адрес
;вначале размер программы в параграфах
       MOV BX,16
       MOV AX,OFFSET PRIVET
       XOR DX,DX
       DIV BX
       INC AX
       MOV BX,AX ;для функции 4AH
;уменьшить размер выделенного пространства
       MOV AH,4AH
       INT 21H ; значение ES совпадает с CS для COM-программы
;найти место для PSP
       MOV BX,16
       MOV AH,48H
       INT 21H
       MOV ES,AX
;пересылка PSP
       MOV SI,0
       MOV DI,0
       MOV CX,100H
       CLD
       REP MOVSB
;установка нового PSP
       MOV AH,50H
       MOV BX,ES
       INT 21H
```

```
;"испортим" старый PSP
      MOV WORD PTR CS:[0],0
      MOV WORD PTR CS:[0AH],0
      MOV WORD PTR CS:[0AH+2],0
;все освободить самому
      MOV AH,49H
      INT 21H
      MOV ES,ES:[2CH]
      MOV AH,49H
      INT 21H
      PUSH CS
      POP ES
      MOV AH,49H
      INT 21H
;выход
      MOV AH,4CH
      INT 21H
PRIVET:
CODE ENDS
      END BEGIN
```

Рис. 11.9. Пример программы, использующей новый PSP.

Чтобы понять суть проблемы, Вам необходимо поэкспериментировать с этой программой. Сделаю несколько пояснений. Если испортить некоторые байты в PSP (см. текст программы), то будет невозможно выйти в DOS даже с помощью функции 4CH. Поэтому перед тем, как испортить PSP, мы переносим его в другое место и при помощи функции 50H сообщаем DOS, что у нас теперь новый PSP. Попробуйте убрать вызов данной функции, и программа перестанет выходить в DOS. Есть еще одна особенность: все занятые блоки памяти приходится освобождать самостоятельно, так как функция 4CH перестает справляться со всеми своими обязанностями. Последнее легко объясняется: освобождение занятой памяти начинается с PSP.

Глава 12. TSR-программы (резидентные).

- Я извиняюсь, - заговорил он подозрительно, - вы кто такой будете? Вы - лицо официальное?
- Эх, Никанор Иванович! - задушевно воскликнул неизвестный. Что такое официальное лицо или неофициальное? Все зависит от того, с какой точки зрения смотреть на предмет, все это, Никанор Иванович, условно и зыбко.

М.А. Булгаков
Мастер и Маргарита.

В данной главе будет рассмотрено явление, которое в операционной системе MS DOS играет роль многозадачности. С истинной многозадачностью мы столкнемся, когда будем рассматривать программирование в операционной системе Windows. Многозадачность же, рассматриваемая в этой главе, может быть названа «нелегальной».

Термин TSR происходит от английской фразы Terminate State Resident, т.е. "завершить и остаться резидентным". Резидентная программа, находясь в памяти, выполняет свои функции через перехваченные прерывания. Связь с резидентной программой также осуществляется посредством прерываний. Часто такие программы играют роль драйверов - программ поддержки, каких-либо внешних устройств. Наиболее часто употребляемые резидентные драйверы обеспечивают русифицированную работу экрана и клавиатуры или поддерживают работу мыши. Популярны резидентные переводчики и различные резидентные детекторы, позволяющие на ранней стадии обнаруживать появление вируса.

Наличие в памяти резидентных программ предполагает как бы некое подобие многозадачного режима, в котором, однако, есть главная задача, выполняющаяся в данный момент. Остальные же задачи получают управление время от времени через перехваченные ими прерывания либо "по милости" главной задачи. Таким образом, операционная система фактически не "знает" об их существовании. Многие авторы говорят в этой связи об изначальной порочности TSR-программ. Не вступая в долгую дискуссию с такими утверждениями, замечу, однако, что:

1. На TSR-программах держится значительная часть как чисто системной, так и прикладной программной поддержки в операционной системе MS DOS.
2. При написании любой программы нужно придерживаться определенных правил, а ошибка в программе приводит к полной и частичной ее неработоспособности.
3. Многие проблемы, рассмотренные в настоящей главе, носят общий характер для любых многозадачных сред.

Ниже мы попытаемся изложить основные положения, которые необходимо знать при написании резидентных программ.

I. Как программа может остаться резидентной.

Существует два легальных документированных способа оставить программу резидентной в памяти. Это прерывание 27H, которое используется для COM-программ, и функция DOS 31H, которая может использоваться как COM так и EXE-программами. Начнем по порядку.

Прерывание 27H.

Вход: DX - адрес первого байта за резидентной частью программы;

CS - на начало PSP.

Отсчет ведется от начала PSP. Если NO_RES - метка начала нерезидентной части программы, то фрагмент, оставляющий программу в памяти, будет следующим:

```
LEA  DX,NO_RES
INT  27H
```

Как видим, все легко и просто. Прерывание 27H стандартным образом возвращает управление DOS с восстановлением векторов 22H, 23H, 24H.

Функция 31H.

AH - 31H,

AL - код выхода,

DX - число параграфов памяти, оставляемых резидентно. Отчет ведется от начала PSP.

Данная функция хороша тем, что размер оставляемого блока для нее не важен. Поэтому ею могут пользоваться EXE-программы. Конечно, в этом случае нужно правильно рассчитать длину оставляемого блока.

Рассмотрим пример (Рис. 12.1). Советую обратить внимание на следующие моменты. В данном примере тип выравнивания сегментов (см. главу 13) - PARA, т.е. сегменты должны начинаться на границе параграфа. Поэтому мы добавляем к длине каждого сегмента 16. Конечно, может оказаться, что мы зарезервируем несколько больше байт, чем нужно (если длина сегмента окажется кратной 16), но зато не потеряем ни одного байта. Второе, мы считаем длину каждого сегмента отдельно, т.к. сумма длин (в байтах) может оказаться больше 64 K. Наконец, мы используем здесь оператор ассемблера SIZE, который определяет длину указанного операнда в байтах. Хотя, естественно, можно было бы обойтись и без него.

Имейте в виду, что, когда запускается программа, ей от родительского процесса передается окружение - область памяти, содержащей строки, помещаемые туда командами SET, PATH, PROMPT. Длина окружения может достигать 32 K. Если Ваша программа останется в памяти резидентной, то вместе с ней в памяти останется и окружение. Ниже мы остановимся на этой проблеме.

```
;сегмент данных
DSEG SEGMENT
        BUFER EB 2000 ШР(?)
        P1 DD ?
DSEG ENDS
```

```
;сегмент
SSEG SEGMENT STACK
ST    DW 100 DUP(?)
SSEG ENDS
;сегмент кода
CODE SEGMENT
      ASSUME CS:CODE, DS:DATA, SS:SSEG
BEGIN:
      .
      .
;все, что ниже в памяти не останется
NO_RES:
      MOV  AX, SIZE BUFER+SIZE P1+16 ;сегмент данных
      MOV  BX, SIZE ST+16 ; сегмент стека
      MOV  DX,NO_RES-BEGIN+16 ; часть сегмента кода
      MOV  CL,4
      SHR  AX,CL    ; делим на 16
      SHR  BX,CL
      SHR  DX,CL
      ADD  AX,BX
      ADD  DX,AX
      ADD  DX,10H   ;учтем PSP - 10H параграфов
      MOV  AX, 3100H
      INT  21H
CODE ENDS
      ЕЮ BEGIN
```

Рис. 12.1. Пример использования функции 31H для EXE-программы.

Итак, мы познакомились с легальными способами того, как программа может быть оставлена памяти. Если Вы наберетесь терпения, то в главе 17 узнаете, как это делают компьютерные вирусы. На Рис. 12.10 представлена резидентная программа, остающаяся в памяти по одному из вирусных способов.

II. Перехват прерываний.

В главе 9 мы довольно подробно говорили о способах перехвата прерываний. Все сказанное, разумеется, справедливо и в случае TSR-программ. Но есть и серьезное отличие - TSR-программа большую часть времени является фоновой задачей. Она не должна мешать работе других программ, запущенных как до нее, так и после, если только это не входит в ее непосредственные функции. Поэтому, если Ваша программа должна перехватывать какое-либо прерывание, позаботьтесь, чтобы программы, которые перехватили это прерывание раньше, также получили управление. Используйте для этого длинные JMP или CALL (см. главу 9).

В том случае, если после вызова прерывания Ваша программа должна еще что-то делать (см. ниже) нужно быть крайне осторожным. Может оказаться, что данное прерывание будет возвращать в регистрах некоторую информацию, которая не должна потеряться. Особенно это касается прерываний 21H, 13H, 25H, 26H, 16H. В случае регистра флагов этот вопрос может быть решен с использованием RET 2 вместо IRET (ниже будут приведены несколько фрагментов). Что касается других регистров, то здесь надо пользоваться простыми и надежными правилами:

1) Перед вызовом прерывания все необходимые регистры должны содержать то же, что они содержали при входе в Вашу процедуру.
2) Перед выходом из Вашей процедуры, эти регистры должны содержать то же, что они содержали при возврате в Вашу процедуру из прерывания.

В случае вызова прерывания через JMP, естественно, будет работать только первый пункт.

В случае аппаратных прерываний, таких, как 8H, 9H, правило значительно упрощается, т.к. эти прерывания не меняют содержимое регистров. В этом случае в начале процедуры следует сохранить содержимое всех используемых Вами регистров, а перед выходом восстановить это содержимое (PUSH AX/POP AX и т.д.).

В заключение этого раздела сошлюсь на Рис. 12.2 и Рис. 12.3, которые иллюстрируют сказанное выше.

```
;процедура обработки прерывания
INT_N PROC FAR
;сохраняем нужные регистры, в том числе и регистр флагов
        PUSHF
        PUSH AX
        .
        .
;выполняем необходимую работу
        .
        .
;восстанавливаем регистры
        .
        .
        ГОР AX
        POPF
;вызываем прерывание
        CALL DWORD PTR CS:OLD_INT_N
;сохраняем нужные регистры
        PUSHF
        PUSH AX
        .
        .
;выполняем необходимую работу
        .
        .
```

```
;восстанавливаем регистры
        .
        .
        ГОР AX
        ГОРЕ
        RETF 2
INT_N   ENDP
```

Рис. 12.2. Схема 1 обработки прерывания резидентной программой.

```
;процедура обработки прерывания
INT_N PROC EAR
;сохраняем нужные регистры, в том числе и регистр флагов
        PUSHF
        PUSH AX
        .
        .
;выполняем необходимую работу
        .
        .
;восстанавливаем регистры
        .
        .
        POP AX
        ГОРЕ
;вызываем прерывание
        JMP DWORD PTR CS:OLD_INT_N
INT_N ENDP
```

Рис. 12.3. Схема 2 обработки прерывания резидентной программой.

III. Как обнаружить себя в памяти.

Если в Вашей программе не реализован механизм обнаружения собственной копии в памяти, то при повторном запуске она снова останется в памяти и т.д. Согласитесь, что это не совсем удобно, а главное непрофессионально. Существует несколько способов обнаружения копии в памяти. Рассмотрим все по порядку:

1. Этот способ я назвал бы так: "внешняя привязка к перехваченному вектору прерывания". Суть этого метода заключается в том, что в теле программы помещается некоторый признак - обычно это слово. Далее делается предположение, что, поскольку резидентной программой был перехвачен определенный вектор, то он должен быть направлен в тело программы. Таким образом, мы получаем адрес входа в процедуру прерывания и по этому адресу ищем признак присутствия. На Рис. 12.4 представлен фрагмент TSR-программы с признаком присутствия. На Рис. 12.5 представлен фрагмент той же программы, определяющий наличие своей копии в памяти. Для опреде-

ленности приведен пример с перехватом прерывания 8. Подчеркну, что оба фрагмента относятся к одной и той же программе.

```
        .
        .
PRIZN   DW AB12H
INT_8   PROC FAR
        .
        .
INT_8   ENDP
```

Рис. 12.4. Фрагмент TSR-программы с признаком присутствия.

```
        .
        .
        MOV AX,3508H
        INT 21H
        CM? ТОЮ PTR ES:[BX-2],PRIZN ;проверяем на присутствие
        .
        .
```

Рис. 12.5. Фрагмент TSR-программы, определяющей свое присутствие в памяти.

Излагаемый метод не слишком совершенен, т.к. после запуска программы другие резиденты могут перехватить ключевое прерывание, и программа перестанет обнаруживать себя в памяти.

2. По аналогии с предыдущим способом назовем этот метод как "внутренняя привязка к перехваченному прерыванию". Как и в предыдущем случае, используется перехваченное программой прерывание. Используется заведомо не существующая функция этого прерывания. В ответ на вызов такой функции программа, находящаяся в памяти, посылает в ответ (в одном из регистров) код, сигнализирующий о своем присутствии в памяти. Замечу, что таким способом можно не только определять присутствие программы в памяти, но и отдавать указания программе, находящейся в памяти. Через один из регистров можно передать сегментный адрес резидента, который необходим для удаления его из памяти. Указанный метод более надежен, прост и универсален, чем предыдущий. Признаюсь, что он мне нравится больше остальных.

В DOS существует прерывание (2FH), которое, по сути, предназначено именно для целей взаимодействия резидентных программ друг с другом, в том числе и для обнаружения себя в памяти. Это прерывание используют утилиты DOS: PRINT, ASSIGN, SHARE. Функции же 80H-FFH предоставлены пользователю (функции, естественно, передаются через AH). Здесь, правда, существует одна не слишком приятная проблема. Могут совпасть номера функций, используемых разными программами (и такие случаи уже были). Выйти из этого положения можно следующим способом: Ваша программа должна реагировать не просто на посланный код, а, скажем,

на последовательность кодов. Тем самым можно свести почти до нуля вероятность совпадений.

Заключая рассказ о данном методе, предлагаю (Рис. 12.6 - 12.7) фрагменты программы, иллюстрирующие это метод. Для определенности взято прерывание 16Н.

```
        .
        .
INT16 PROC
        CMP AH,20H
        JNZ CONT
        MOV AX,789AH        ;код присутствия
        MOV BX,CS           ;сегментный адрес
        IRET
CONT:
        JMP DWORD PTR CS:OLD16
INT16 ENDP
        .
        .
```

Рис. 12.6. Фрагмент, передающий код возврата.

```
        .
        .
        MOV AH,20H
        INT 16H
        CMP AX,789AH
        JZ YES          ;программа в памяти
        .
        .
```

Рис. 12.7.Фрагмент, определяющий наличие программы в памяти.

3. Данный метод наиболее сложен, но и при правильной его реализации, это самый надежный подход. В конце главы будет приведена программа, в которой этот метод будет реализован.

В главе 2 сообщалось о блоках памяти, которыми оперирует DOS. Резидентной программе также отводится свой блок. Метод обнаружения резидентной программы в памяти заключается в том, чтобы сканировать все занятые блоки на предмет обнаружения программы. Сканировать можно все блоки подряд, но можно и сделать это более аккуратно. Оказывается, в заголовке блока содержится параграф владельца. В большинстве случаев это PSP программы. Причем PSP может находиться в другом блоке памяти. Вот здесь и надо провести проверку. В комментарии к примеру на Рис. 12.8 сказано об этом более подробно. Начало PSP можно обнаружить по команде INT 20H, стоящей вначале.

IV. Активизация резидентной программы/Проблемы неинтерабельности.

Щелкни кобылу в нос - она махнет хвостом.

Козьма Прутков.

Многие резидентные программы предполагают активизацию при нажатии некоторой группы клавиш или наступления иного события. Под активизацией мы понимаем появление меню или выполнение какой-либо операции. Проблема совсем не простая, как может показаться на первый взгляд. Представьте, что с диском работает некоторая программа, и в этот момент происходит активизация резидентной программы. Причем резидентная программа также предполагает работу с диском. Согласитесь, что даже из общих соображений здесь не все гладко.

Более четко (несколько сузив) эту проблему можно сформулировать следующим образом: можно ли во время выполнения какой-либо функции DOS снова запустить функцию DOS? Такая ситуация может возникнуть как раз в том случае, если TSR-программа активизировалась во время работы функции DOS. Если TSR-программа не использует функции DOS, то проблемы нет (есть другие - о них будет сказано позже). MS DOS — неинтерабельная операционная система, т.е. не позволяет запускать себя из себя же. В основном это связано с тем, что DOS устанавливает свой стек, и если войти в DOS вторично, то вторично будет установлен тот же стек и будет использоваться та же область данных - последствия этого очевидны.

Если Ваша резидентная программа предполагает использование функций DOS, то перед тем, как это сделать, необходимо проверить внутренний флаг DOS. Это можно сделать при помощи недокументированной функции DOS 34H [31] Это функция возвращает в ES:[BX] указатель на байт, который равен 1, если выполняется какая-либо функция DOS, и 0 в противном случае. Проверяя этот флаг, Ваша программа должна дождаться, когда он установится в 0 и только тогда выполнить необходимую операцию или выдать меню (если затем предполагается выполнение "опасных" операций). Использование данного метода не всегда эффективно, т.к. некоторые программы ожидание нажатия клавиши осуществляют посредством функций DOS, т.е. флаг в течение этого ожидания будет равен 1.

Другой подход предполагает перехват и отслеживание прерывания 21H. При этом, перед тем как выполнять какую-либо функцию символьного ввода, следует проверять буфер клавиатуры и вызывать эту функцию только при появлении символа в буфере клавиатуры. На Рис. 12.8 представлен фрагмент программы, демонстрирующий сказанное. Обращаю Ваше внимание, что отслеживается функция OAH буферизированного ввода. Проверка буфера осуществляется при помощи прерывания 16H. Задержка делается для получения безопасного "окна", в котором может быть вызвана резидентная программа. Заметьте, что выход из прерывания осуществляется по RETF 2. При FLAG_21=0 TSR-программа может быть активизирована.

---

[31] Сама эта функция безопасна для выполнения.

Иногда для проверки наличия в буфере символа используют функцию OBH.

```
INT21 PROC FAR
        STI
        CMP  AH,0AH
        JZ   A1Z
        JMP  SHORT A2ZA1Z:
        PUSH CX
        PUSH AX
A3Z:
        MOV  CX,0FFFFH
A4Z:
        MOV  AH,1
        LOOP A4Z
        INT  16H
        JZ   A3Z
        POP  AX
        POP  CX
A2Z:
        MOV  CS:FLAG_21,1
        PUSHF
        CALL DWORD PTR CS: [OFF_21]
        MOV  CS:FLAG_21,0
        RETF 2
INT21 ENDP
```

Рис. 12.8. Отслеживание прерывания 21H.

Активизация резидентной программы часто необходима при нажатии каких-либо клавиш. Это нажатие можно отследить при помощи прерывания клавиатуры. Однако часто активизировать программу в данный момент не желательно по причине безопасности. Обычно поступают следующим образом. По нажатию горячих клавиш устанавливается флаг активизации программы. Процедура, обрабатывающая прерывание таймера, проверяет этот флаг. Если флаг установлен, то проверяется флаг 21-го прерывания (а также других, см. ниже), и если не работают функции этого прерывания, то программа активизируется. Можно вообще не использовать прерывание клавиатуры, а проверять периодически, нажата или нет та или иная клавиша из прерывания по времени.

В качестве еще одной точки входа можно использовать прерывание 28H. Известно, что DOS вызывает это прерывание перед вызовом функций символьного ввода-вывода. Причем, когда вызывается это прерывание, система работает в безопасном режиме и можно активизировать резидентную программу. Как использовать это прерывание — см. Рис. 12.9 и комментарий к нему.

Для повышения безопасности Вашей программы следует, кроме 21-го прерывания, отслеживать еще 13-е, 25-е и 26-е прерывания. Эти прерывания также небезопас-

ны и могут вызываться некоторыми программами, минуя функции DOS (см. Рис. 12.9 и комментарий к нему).

Таким образом, можно говорить о синхронной и асинхронной активизации резидентной программы. Синхронная активизация осуществляется в том случае, если Вы не собираетесь использовать функции DOS. При этом ничего отслеживать не надо и можно активизироваться сразу по нажатию клавиш.

V. Использование памяти.

TSR-программа, как и любая другая, может запрашивать у системы память. Однако TSR-программа должна выполнять свои функции и во время работы других программ. В этом случае с большой долей вероятности у системы может не оказаться свободной памяти. Как поступить в этом случае? Здесь возможны два пути.

1. Резервировать память внутри программы либо резервировать во время установки программы в памяти. Если памяти требуется не очень много, то такой подход допустим и даже является предпочтительнее второго.

2. Использовать внешнюю память. Вместо того чтобы писать данные в память, они записываются на диск. Но как быть в том случае, если обмен данных должен происходить быстро? Выберете область памяти, которая Вам удобна (только не системную) и которая была свободна при запуске программы. При активизации TSR-программа должна сбросить эту область памяти на диск. Теперь она может использовать память по своему усмотрению. При выходе из активного состояния область памяти должна быть восстановлена.

Этот весьма заманчивый подход, однако, весьма опасен. Дело в том, что выбранную область памяти может занимать программа (запущенная после Вашей), которая перехватит некоторые векторы, а процедуры обработки этих прерываний поместит как раз в эту область. Если при активной работе Вашей программы будет вызвано именно это прерывание, то катастрофы не избежать. Что можно здесь посоветовать? Предусмотрите процедуры обработки для всех векторов, которые используются при активной работе программы. Запомните их значения при установке программы в памяти. При активизации программа должна перенаправить эти вектора на Ваши процедуры. Таким образом Вы исключите процедуры обработки прерываний, которые были установлены после запуска программы. При выходе из активного состояния, векторы прерываний следует восстановить.

3. Использовать дополнительную и расширенную память. В главе 22 будет подробно рассказано об этих видах памяти. Разумеется, и здесь возникает проблема совместного использования памяти несколькими программами. При запуске резидентная программа может зарезервировать часть дополнительной или расширенной памяти, а затем при активизации использовать ее.

VI. Конфликтные ситуации.

Рассмотрим некоторые конфликтные ситуации, которые могут возникнуть при работе с резидентными программами.

1.Хорошая резидентная программа должна предусматривать удаление себя из памяти (см. Рис. 12.9 и комментарий к нему). При удалении программы, естественно, следует

восстановить все векторы прерываний. Однако к моменту удаления программы из памяти, некоторые из перехваченных векторов окажутся перехваченными другими программами. Восстановив старые значения векторов, можно исключить те обработчики прерываний, которые были установлены после Вашей программы. Если теперь программы, которые установили эти обработчики, будут восстанавливать значения векторов, то они окажутся направленными в область памяти, где уже нет Вашей программы. Возможна и другая ситуация. К моменту, когда запускалась Ваша программа, были установлены некоторые резидентные программы, которые перехватили некоторые векторы. Затем эти же векторы были перехвачены Вашей программой. После чего резидентные программы, запущенные до Вашей программы, были удалены из памяти. И в том, и другом случаях значения векторов изменились по сравнению с начальными их значениями. Хорошая программа перед очисткой памяти и восстановлением векторов должна по крайней мере предупредить об изменившихся векторах и предложить выбор [32]. Идеальным случаем было бы отслеживание всех изменений векторов вместе с причинами этих изменений. Например, если программа была запущена до Вашей программы, а затем восстановила векторы, то Ваша программа при удалении ее из памяти эти векторы изменять не должна. Проблему контроля измененийй векторов можно решить, отслеживая функции 49H и 4AH и проверяя, не были направлены в освобождаемую область какие-либо векторы.

2. Будьте аккуратны со стеком. Если Ваша программа использует операции со стеком (к таковым относится и вызов процедуры), то лучше использовать свой стек, т.к. памяти может не хватить. Активизация Вашей программы может произойти в момент работы другой, при этом стек последней может быть в данный момент на исходе. Лично я для стека использую PSP, что удобно и вполне достаточно (см. Рис. 12.9). Впрочем, PSP можно использовать и более рационально. Об этом будет сказано ниже.

Здесь есть проблема интерабельности уже по отношению к Вашим процедурам. Если какая-либо Ваша процедура при активизации устанавливает свой стек, то следует отдавать себе отчет о том, что произойдет, когда при работе этой процедуры она будет вызвана вторично. Вы должны предусмотреть определенный механизм многократного выделения стека либо запретить вторичный вход в процедуру (см. Рис. 12.9).

3. Активизация резидентной программы может произойти во время работы другой. Некоторые программы (к таким относятся многие программы фирмы Микрософт) перехватывая прерывание клавиатуры, реагируют на нажатие клавиш непосредственно через это прерывание. Если активизация резидентной программы предполагает работу с меню, то следует еще раз перенаправить вектор прерывания клавиатуры на Вашу процедуру обработки. Если этого не сделать, то Ваше меню не будет нормально работать или же произойдет "зависание". При выходе из активного состояния вектор, естественно, следует восстановить.

4. Если резидентную программу предполагается активизировать только при определенном режиме экрана (например, текстовом или определенном графическом), то перед активизацией следует проверить режим экрана и среагировать должным образом.

5. Если при активизации Вашей резидентной программы предполагается открывать файлы, то новые описатели будут, естественно, помещаться в PSP той програм-

[32] Этого часто не делают даже довольно уважаемые программы.

мы, которая работала на момент активизации (см. главу 8). В этом случае может возникнуть маловероятная ситуация, когда описателей просто не хватит. Решить данную проблему можно следующим образом: получить адрес PSP прерванной задачи и сохранить его (функция DOS 62H). Установить адрес своего PSP (функция DOS 50H недокументирована). Перед выходом из программы следует восстановить адрес старого PSP (функция 50H).

6. Также маловероятна (но возможна) следующая ситуация. Некоторые программы после выполнения функции DOS вызывают функцию 59H для получения расширенного кода ошибки. Резидентная программа может активизироваться перед вызовом этой функции. Если она также использует какие-то функции DOS, то код расширенной ошибки может измениться. Чтобы этого не случилось, следует предварительно получить код при помощи функции 59H. Перед выходом же из активного состояния код расширенной ошибки должен быть восстановлен (функция 5DH, подфункция 5 легализована в DOS 5.0).

7. Проблема неинтерабельности MS DOS. О том, как решить данную проблему, мы уже говорили. Можно использовать и другой подход. Если бы мы знали, где расположена область системных стеков и ее размер, то могли бы сохранить эту область перед вызовом функций DOS, а затем при выходе из резидентной программы восстановить эту область. Данные об этой области можно получить при помощи функции 5DH (подфункция 6).

Вызов:

```
MOV AX,5D06H
INT 21H
```

Выход:
DS:SI - начало области
CX - размер области

VII. Пример резидентной программы.

Здесь приводится пример резидентной программы, которая, возможно, даже окажется для Вас полезной. Эта программа заносит копию экрана в файл SCREEN.TXT в текущей директории [33]. При вторичном запуске программа удаляет свою резидентную копию из памяти. Рассмотрим теперь основные моменты, на которые, на мой взгляд, следует обратить внимание.

1. Обнаружение резидентной копии в памяти осуществляется посредством просмотра занятых блоков памяти (процедура SCAN). Предполагается, что PSP начинается сразу после заголовка блока. Конечно, блок может быть занят и окружением, но просмотр лишних блоков не мешает программе выполнять свои функции. Поиск первого MCB осуществляется через функцию 52H прерывания 21H.

2. Перед тем как остаться резидентной, программа освобождает блок памяти, занятой окружением (процедура FREE_ENV).

[33] То, что в [7] приводится пример программы, выполняющей похожие функции, - чистое совпадение. С этой статьей я познакомился уже после написания программы в данной главе. Впрочем, легко увидеть, что программы совершенно разные.

3. При восстановлении векторов их старые значения находятся по смещению от начала PSP (процедура RE_VECT). Замечу, что эти смещения фактически совпадают со значением соответствующих меток. Используя числовые смещения, я хотел подчеркнуть лишний раз, что старые значения векторов берутся из резидентного модуля.

4. Обратите внимание на процедуры обработки прерываний. Особенно это касается прерываний 25H и 26H. Они оставляют в стеке лишнее слово. Этим объясняется появление в этих процедурах команды POP SI. Не страшно, что будет испорчено содержимое SI, ведь в руководстве сказано что, прерывания 25H/26H портят содержимое всех регистров, кроме DS, ES, SS, SP.

5. В программе практически не используется стек процесса, из которого активизируется программа. Программа устанавливает свой стек на конец PSP (старшие адреса).

6. Копирование экрана производится в два приема. Вначале делается мгновенный снимок в буфер, находящийся внутри программы. Сброс же буфера в файл происходит в безопасный момент либо из прерывания по времени, либо 28-го прерывания.

```
CODSEG  SEGMENT
ASSUME  CS:CODSEG
        ORG 100H
BEGIN:
        JMP BEG
PRIZN   DW  0F034H ; смещение 103H от начала PSP
; старые векторы прерывания
                   ;смещение от начала PSP
OLD09O DW ?        ;+105H
OLD09S Ш  ?        ;+107H
OLD08O Ш  ?        ;+109H
OLD08S Ш  ?        ;+10BH
OLD21O DW ?        ;+10DH
OLD21S да ?        ;+10FH
OLD13O да ?        ;+111H
OLD13S да ?        ;+113H
OLD25O да ?        ;+115H
OLD25S да ?        ;+117H
OLD26O да ?        ;+119H
OLD26S ОТ ?        ;+11BH
OLD28O DW ?        ;+11DH
OLD28S да ?        ;+11FH
OLD24O Да ?
OLD24S ОТ ?
OLD23O Да ?
OLD23S Да ?
;-------------------------
P1      DB  0      ;запрос был
P2      DB  0      ;буфер готов
```

```
P3      DB  O   ;признак обработки
;признаки работы прерываний
P21     DB  0   ; 21H
P13     DB  0   ; 13H
P25     EB  0   ; 25H
P26     DB  0   ; 26H
;буфер для хранения копии экрана
BUFER   DB  2050 DUP(?)
;для временного хранения регистров
_AX     ВД  ?
_ES     Ш   ?
_SS     Ш   ?
_SP     Ш   ?
;имя файла для записи копии экрана
FILE    DB "SCREEN.TXT",0
;область резидентных процедур прерывания клавиатуры
INT09 PROC FAR
        CMP CS:P1,1  ;запрос был
        JZ YES
        MOV  CS:_AX,AX
        MOV  CS:_ES,ES
        IN   AL,60H
        CMP  AL,15   ;не TAB ли?
        JZ   TAB
        MOV  AX,CS:_AX
        JMP  SHORT YES
TAB:
        MOV  AX,0
        MOV  ES,AX
        TEST BYTE PTR ES:[417H],4
        JZ   NO_CTRL
        MOV  CS:P1,1  ;делаем запрос на выполнение операции
NO_CTRL:
        MOV  AX,CS:_AX
        MOV  ES,CS:_ES
YES:
        JMP DWORD PTR CS: [OLD09O]
INT09 ENDP
;прерывание таймера
INT08 PROC FAR
;выполняем стандартную операцию
        PUSHF
        CALL DWORD PTR CS: [OLD08O]
        CMP  CS:P3,1
```

```
        JNZ    NO_WORK
        JMP    WORK
NO_WORK:
        MOV  CS:P3,1
;устанавливаем свой стек, направленный на PSP
        MOV  CS:_AX,AX
        MOV  CS:_SS,SS
        MOV  CS:_SP,SP
        MOV  AX,CS
        MOV  SS,AX
        MOV  SP,100H
        MOV  AX,CS:_AX
;сохраняем регистры в новом стеке
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH DX
        PUSH SI
        PUSH DI
        PUSH ES
        PUSH DS
        CMP  CS:P1,1      ;был ли запрос
        JNZ  N01
        CMP  CS:P2,1      ;не готов ли буфер
        JZ   NO2
        CALL VID_BUF      ;копируем экран в буфер
        MOV  CS:P2,1
NO2:
;проверка флагов
        CMP  CS:P13,1
        JZ   NO1
        CMP  CS:P25,1
        JZ   NO1
        CMP  CS:P26,1
        JZ   N01
        CMP  CS:P21,1
        JZ   NO1
        CALL BUF_DISK      ;сбрасываем буфер на диск
        MOV  CS:P1,0
        MOV  CS:P2,0
N01:
;восстанавливаем регистры
        POP  DS
        ЮP   ES
```

```
        ЮР   DI
        ЮР   SI
        ЮР   DX
        ЮР   CX
        ЮР   BX
        ЮР   AX
;восстанавливаем старый стек
        MOV  SS,CS:_SS
        MOV  SP,CS:_SP
        MOV  CS:P3,0
WORK:
        IRET
INT08 ENDP
;прерывание 21
INT21 PROC FAR
        STI
        MOV CS:P21,1
        PUSHF
        CALL DWORD PTR CS: [OLD21O]
        MOV CS:P21,0
        RETF 2
INT21 ENDP
;прерывание 13
INT13 PROC EAR
        MOV CS:P13,1
        PUSHF
        CALL DWORD PTR CS:[OLD13O]
        MOV CS:P13,0
        RETF 2
INT13 ENDP
;прерывание 25
INT25 PROC FAR
        MOV CS:P25,1
        PUSHF
        CALL DWORD PTR CS: [OLD25O]
        POP SI          /убираем лишнее слово
        MOV CS:P25,0
;при выходе из процедуры одно лишнее слово остается в стеке
        RETF
INT25 ENDP
/прерывание 26
INT26 PROC FAR
        MOV CS:P26,1
        PUSHF
```

```
        CALL DWORD PTR CS:[OLD26O]
        ЮР SI           ; убираем лишнее слово
        MOV CS:P26,0
;при выходе из процедуры, одно лишнее слово остается в стеке
        RETF
INT26 ENDP
;прерывание 28 - еще одна точка входа в процедуру BUF_DISK
INT28 PROC FAR
        PUSHF
        CALL СТОЮ PTR CS:[OLD28O]
        CMP CS:P2,1
        JNZ  NET
        MOV  CS:P3,1
        CALL BUF_DISK
        MOV  CS:P1,0
        MOV  CS:P2,0
        MOV  CS:P3,0
NET:
        IRET
INT28 ENDP
;обработчик критических ошибок
INT24 PROC FAR
        MOV AL, 3
        IRET
INT24 ENDP
;обработчик прерывания 23H
INT23 PROC FAR
        IRET
INT23 ENDP
;другие процедуры
;из видеопамяти в буфер
VID_BUF PROC
;копируем экран в буфер
        CLI
        LEA  DI,CS:BUFER
        XOR  SI, SI
        MOV  AX,0B800H
        MOV  ES,AX
        MOV  CX,2000
        XOR  BL,BL
LOO2:
        MOV  AL,ES:[SI]
        MOV  CS:[DI],AL
        INC  DI
```

```
        ADD  SI,2
        INC  BL
;в конце строки 13,10
        CMP  BL, 80
        JNZ  N03
        XOR  BL,BL
        MDV  BYTE PTR CS:[DI],13
        INC  DI
        MOV  BYTE PTR CS:[DI],10
        INC  DI
NO3:
        LOOP LOO2
        STI
        RET
VID_BUF ENDP
;из буфера на доек
BUF_DISK PROC
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH DX
        PUSH ES
        PUSH DS
        PUSH CS
        ЮР   DS
;устанавливаем обработчик критических ошибок
        MDV  AX,3524H
        INT  21H
        MDV  CS:OLD24S,ES
        MDV  CS:OLD24O,BX
        MDV  AX,2524H
        LEA  DX,CS:INT24
        INT  21H
        MOV  AX,3523H
        INT  21H
        MDV  CS:OLD23S,ES
        MDV  CS:OLD23O,BX
        MDV  AX,2523H
        LEA  DX,CS:INT23
        INT  21H
; открываем файл (FILE)
        MDV  AH,3CH
        LEA  DX,CS:FILE
        XOR  CX,CX
```

```
        ЮТ    21H
        JC    EX
        MOV   BX,AX
; записываем в файл содержимое экрана
        LEA   DX,CS:BUFER
        MOV   AH,40H
        MOV   CX,2050
        ЮТ    21H
;закрываем файл
        MOV   AH,3EH
        ЮТ    21H
EX:
;восстанавливаем обработчик критических ошибок
        MOV   AX,2524H
        MOV   DX,CS:OLD24O
        MOV   DS,CS:OLD24S
        INT   21H
        MOV   AX,2523H
        MOV   DX,CS:OLD23O
        MOV   DS,CS:OLD23S
        INT   21H
        ТОР   DS
        ЮР    ES
        ТОР   DX
        POP   CX
        ЮР    BX
        ЮР    AX
        RET
BUF_DISK ENDP
RES_END:
;не резидентная часть
;сканирование памяти
;если программа найдена, то в AX - 1, иначе AX - O
;ES - сегмент, где находится программа
SCAN PROC
        PUSH  BX
        PUSH  DS
        PUSH  DX
        PUSH  SI
;в SI сегмент работающей программы (не резидентной)
        MOV   SI,CS
;находим первый блок
        MOV   AH,52H
        ЮТ    21H
```

```
        MOV ES,ES:[BX-2]
        MOV  AX,1
LOO:
        CMP  WORD PTR ES:[1],0
        JZ   CONT5
        PUSH ES
        POP  BX
        INC  BX
;не данный ли это сегмент?
        CMP SI,BX
        JZ   CONT5
        MOV  DS,BX
;нет ли резидента в сегменте
        MOV  BX,CS:PRIZN
        CMP ТОЮ PTR DS:[103H],BX
        JZ   _END
CONT5:
;не последний ли это блок ?
        CMP  BYTE PTR ES:[0],'M'
        JZ   CONT4
        XOR  AX,AX
        JMP  SHORT _END
CONT4:
;находим адрес заголовка следующего блока
        MOV  DX,ES:[3]
        MOV  BX,ES
        ADD  BX,DX
        INC  BX
        MOV  ES,BX
        JMP  SHORT LOO
_END:
        MOV  BX,ES
        INC  BX
        MOV  ES,BX
        ГОР  SI
        ГОР  DX
        ГОР  DS
        ГОР  BX
        RET
SCAN ENDP
;установить векторы
SET_VECT PROC
        PUSH AX
        PUSH BX
```

```
        PUSH  DX
        PUSH  ES
;сохраним старые векторы
        MOV   AX,3509H
        INT   21H
        MOV   CS:OLD09S,ES
        MOV   CS:OLD09O,BX
        MOV   AX,3508H
        ЮТ    21H
        MOV   CS:OLD08S,ES
        MOV   CS:OLD08O,BX
        MOV   AX,3521H
        ЮТ    21H
        MOV   CS:OLD21S,ES
        MOV   CS:OLD21O,BX
        MOV   AX,3513H
        ЮТ    21H
        MOV   CS:OLD13S,ES
        MOV   CS:OLD13O,BX
        MOV   AX,3525H
        ЮТ    21H
        MOV   CS:OLD25S,ES
        MOV   CS:OLD25O,BX
        MOV   AX,3526H
        ЮТ    21H
        MOV   CS:OLD26S,ES
        MOV   CS:OLD26O,BX
        MOV   AX,3528H
        ЮТ    21H
        MOV   CS:OLD28S,ES
        MOV   CS:OLD28O,BX
;установим векторы
        MOV   AX,2509H
        LEA   DX, ЮТ09
        ЮТ    21H
        MOV   AX,2508H
        LEA   DX,INT08
        ЮТ    21H
        MOV   AX,2521H
        LEA   DX,INT21
        ЮТ    21H
        MOV   AX,2513H
        LEA   DX,                 ЮТ13
        ЮТ    21H
```

```
        MOV  AX,2525H
        LEA  DX,INT25
        INT  21H
        MOV  AX,2526H
        LEA  DX,INT26
        INT  21H
        MOV  AX,2528H
        LEA  DX,INT28
        INT  21H
;---
        ГОР  ES
        ГОР  DX
        ГОР  BX
        ГОР  AX
        RET
SET_VECT ENDP
;восстановить векторы
RE_VECT PROC
        PUSH AX
        PUSH DX
        PUSH DS
        MOV  AX,2509H
        MOV  DX,WORD PTR ES:[105H]
        MOV  DS,WORD PTR ES:[107H]
        INT  21H
        MOV  AX,2508H
        MOV  DX,WORD PTR ES:[109H]
        MOV  DS,WORD PTR ES:[10BH]
        INT  21H
        MOV  AX,2521H
        MOV  DX,WORD PTR ES:[10DH]
        MOV  DS,WORD PTR ES:[10FH]
        INT  21H
        MOV  AX,2513H
        MOV  DX,WORD PTR ES:[111H]
        MOV  DS,WORD PTR ES:[113H]
        INT  21H
        MOV  AX,2525H
        MOV  DX,WORD PTR ES:[115H]
        MOV  DS,WORD PTR ES:[117H]
        INT  21H
        MOV  AX,2526H
        MOV  DX,WORD PTR ES:[119H]
        MOV  DS,WORD PTR ES:[11BH]
```

```
        INT   21H
        MOV   AX,2528H
        MOV   DX,WORD PTR ES:[11DH]
        MOV   DS,WORD PTR ES:[11FH]
        INT   21H
        POP   DS
        ЮР    DX
        ЮР    AX
        RET
RE_VECT ENDP
;освободить память
ERASE  PROC
        MOV AH,49H
        INT 21H
        RET
ERASE  ENDP
;освободить окружение
FREE_ENV PROC
        PUSH AX
        PUSH  ES
        MOV ES,DS:[2CH]
        CALL  ERASE
        POP ES
        ЮР  AX
        RET
FREE_ENV ENDP
;основная программа
BEG:
        CALL  SCAN
        CMP AX,0
        JZ   NO_PR
        CALL  RE_VECT
        CALL  ERASE
        LEA    DX,TEXT2
        MOV   AH,9
        INT    21H
        RET
NO_PR:
        CALL  SET_VECT
        CALL  FREE_ENV
        LEA  DX,TEXT1
        MDV AH,9
        INT  21H
        LEA  DX,RES_END
        INT  27H
```

```
TEXT1   DB 'Программа установлена в память.',13,10,'$'
TEXT2   DB 'Программа удалена из памяти.',13,10,'$'
CODSEG  ENDS
        END BEGIN
```

Рис. 12.9. Законченный пример резидентной программы. Копирование экрана по CTRL+TAB.

VIII. Еще одна резидентная программа.

Данная резидентная программа с точки зрения ее применения не представляет особого интереса. Она перехватывает прерывание 17H и перенаправляет весь вывод на принтер в файл C:\FILE.TXT. Причем работает она лишь с теми программами, которые используют для вывода непосредственно прерывание 17H. Те программы, которые делают это через DOS'овские функции, работать с ней не будут вследствие неинтерабельности DOS. Я не ствил задачей преодолеть эту проблему. Задача была другой. Я хотел лишь показать, как в принципе можно освобождать PSP программы, оставляя в памяти только код.

```
CODE SEGMENT
        ASSUME CS:CODE
        ORG 100H
BEGIN:
        JMP UST
DES   DW ?
PATH DB 'C:\FILE.TXT',0
BUF   DB  ?
PR    DB  0
OLD_VEC   DW ?
          DW ?
INT17 PROC
;блок подачи сигнала о наличии программы в памяти
        CMP AH,78H
        JNZ PAR
        MOV AH,80H
        MOV DX,CS
        ADD DX,16  ;(!) Вы поняли?
        LEA BP,CS:PR
        MOV SI,CS:OLD_VEC
        MOV DI,CS:OLD_VEC+2
        IRET
PAR:
        PUSH DS
        PUSH BX
        PUSH CX
        PUSH DX
        CMP CS:PR,0
```

```
        PUSH AX
        JNZ DDD
; создать при первом обращении
        MOV AH,3CH
        MOV CX,0
        PUSH CS
        ЮР   DS
        LEA  DX,CS:PATH
        INT  21H
        MOV  CS:PR,1
        MOV  CS:DES,AX
DDD:
;здесь обычная обработка
        POP AX
        CMP AH,0
        JNZ  KON ;нет посылаемого символа
;запись
        MOV  AH,40H
        MOV  CX,1
        MOV  CS:BUF,AL
        MOV  BX,CS:DES
        LEA  DX,CS:BUF
        PUSH CS
        POP  DS
        INT  21H
        JC   OK1
        CMP  AX,CX
        JZ   KON
OK1:
;открыть, если файл был закрыт
        MOV  AL,2
        MOV  AH,3DH
        LEA  DX,DS:PATH
        INT  21H
;указатель на конец
        MOV  CS:DES,AX
        MOV  BX,AX
        MOV  AH,42H
        MOV  CX,0
        MOV  DX,0
        MOV  AL,2
        INT  21H
;запись
        MOV  AH,40H
        MOV  CX,1
```

```
        LEA   DX,CS:BUF
        INT   21H
KON:
        MDV AH,144
        POP DX
        TOP CX
        POP BX
        TOP DS
        IRET
INT17 ENDP
UST:
;не резидентная часть
        MDV AH,78H
        INT 17H
        CMP AH,80H
        JNZ DAL
;удалить программу из памяти
CLOSE:
        MDV ES,DX
        PUSH ES
        XOR AX,AX
        MDV ES,AX
        CLI
        MDV ES:[17H*4],SI
        MDV ES:[17H*4+2],DI
        ГОР ES
        MOV AH,49H
        INT 21H
        LEA DX,TEXT2
        MOV AH,9
        INT 21H
;выход
        MDV AH,4CH
        INT 21H
DAL:
        MOV PR,0
;освободить окружение
        MOV ES,DS:[2CH]
        MDV AH,49H
        INT 21H
;установить вектор 17H
        XOR AX,AX
        MDV ES,AX
        MDV AX,ES:[17H*4]
        MDV CS:OLD_VEC,AX
        MDV AX,ES:[17H*4+2]
```

```
        MDV CS:OLD_VEC+2,AX
        MDV AX,OFFSET INT17
        CLI
        MDV ES:[17H*4],AX
        MDV ES:[17H*4+2],CS
;найти сегментный адрес
        MDV BX,16
        MDV AX, OFFSET _END
        XOR DX,DX
        DIV BX
        INC AX
        MOV BX,AX
        PUSH CS
        ГОР ES
;уменьшить размер выделенного пространства
        MDV AH,4AH
        INT 21H
;найти место для PSP
        MOV BX,16
        MOV AH,48H
        INT 21H
        MDV ES,AX ;сегментный адрес нового PSP
;пересылка PSP
        MOV SI,0
        MDV DI,0
        MDV CX,100H
        CLD
        REP MDVSB ; пересылка
;указать новый PSP
        MOV AH,50H
        MDV BX,ES
        INT 21H
;освободить блок нового PSP
        MOV AH,49H
        INT 21H
;разобраться с блоками памяти
;первый блок
        MOV BX,DS
        DEC BX
        MDV DS,BX
        MDV WORD PTR DS:[1],0     ;блок свободный
        MDV AX,DS:[3]
        PUSH AX
        MOV TORD PTR DS:[3],15 ;это блок всего 240 байт
```

```
,• теперь второй блок
        ADD BX,16
        MOV DS,BX
        INC BX
        MOV BYTE PTR DS:[0],'M' ;заголовок блока
        MOV WORD PTR DS:[1],BX  ;сегментный адрес блока
        MOV  AX,OFFSET CS:UST
        SUB  AX,256
        MOV CX,16
        XOR  DX,DX
        DIV  CX
        INC  AX
        MOV DS:[3],AX           ;размер блока
        MOV  DX,DS
        ADD  DX,AX
        INC  DX
        MOV  DS,DX
        ЮР   DX
        SUB  DX,16
        SUB  DX,AX
        SUB DX,1
;теперь третий блок
        MOV  BYTE PTR DS:[0],'M'
        MOV  WORD PTR DS:[1],0  ;блок свободен
        MOV  DS:[3],DX          /размер блока
;выдать сообщение
        PUSH CS
        POP  DS
        LEA DX,TEXT1
        MOV AH,9
        INT 21H
;выйти обычным образом
        MOV AH,4CH
        INT 21H
TEXT1 DB 'Программа установлена в память.',13,10,'$'
TEXT2 DB 'Программа удалена из памяти.',13,10,'$'
_END:
CODE ENDS
        END BEGIN
```

*Рис. 12.10. Пример **TSR-программы**, освобождающей свой PSP.*

Поясню подробнее, как работает данная программа.

1. Как и **обычно**, программа имеет резидентную и нерезидентную части. Резидентная часть заканчивается меткой UST.

2. Резидентная часть фактически является процедурой обработки прерывания 17H. Вся информация, идущая через это прерывание, выводится в файл C:\FILE.TXT. В регистре AH возвращается байт готовности принтера. Переменная PR содержит признак того, открыт файл FILE.TXT или нет. Нюанс, однако, заключается в том, что если мы выходим из программы, которая ранее выводила данные на принтер, то файлы автоматически закрываются. Поэтому мной предусмотрена простая проверка того, открыт в действительности файл или нет. Если нет, то файл снова открывается и указатель переносится в конец.
3. Через прерывание 17H программа также узнает, загружена ее резидентная часть в память или нет. При этом через регистры возвращается необходимая информация для выгрузки программы из памяти.
4. Данная программа занимает в памяти всего 192 байта. Такая компактность достигается удалением из памяти PSP. Это наиболее сложная часть программы. Изложу пошаговый алгоритм.
   а) "Обрезаем" программу по метке _END, т.е. освобождаем память за программой.
   б) Выделяем блок памяти в 256 байт для нового PSP.
   в) Копируем PSP в этот блок.
   г) Указываем на новый PSP (функция 50H).
   д) Теперь освобождаем этот блок.
   е) Далее наиболее тонкая работа. Приходится работать на уровне блоков MCB. Участок памяти PSP делается свободным. Затем устанавливается блок, в котором будет находиться резидентная часть программы. Наконец объявляется свободной оставшаяся часть программы.
   ж) Выход осуществляется обычным способом - через функцию 4CH. При этом, естественно, блоки, выделенные не через функции DOS, не освобождаются.
5. Создание нового PSP, вообще говоря, не обязательно. Но такая технология понадобится, если Вы захотите хранить в PSP данные (см. также главу 11, Рис. 11.9) или остаться резидентным по другой схеме: сдвигом кода программы в область PSP.

В виде послесловия к данной главе.

Часто применение TSR-программ может быть весьма неожиданным. Лет 10 назад автору попалась программа, не желающая нормально работать с EGA-монитором. Суть проблемы заключалась в том, что портился русский шрифт. Причем на VGA-адаптере все шло нормально. Не имея времени детально разбираться в том, почему так происходит, я написал небольшую резидентную программу, которая по нажатию определенной клавиши инициализировала экран через INT 10H (т.е. происходила перезагрузка шрифтов), а затем удаляла себя из памяти. Проблема была решена за 30 минут.

Изложенная выше теория постепенно теряет свою актуальность. В многозадачных операционных системах все задачи являются равноправными, и обслуживание их берет на себя операционная система. Мое изложение являет собой попытку внести многозадачность в однозадачную операционную систему. Данный материал может быть полезен и в том случае, если Вы захотите разобраться в проблеме построения многозадачной операционной системы.

8 - 4072

Глава 13. Модульное программирование и структура программ.

Не в совокупности ищи единства, но более - в единообразии разделения.

Козьма Прутков.

Мы рассматриваем чисто техническую сторону модульного программирования - разбиение программы на части, раздельное их ассемблирование, затем сборку их с помощью редактора связей. Такой подход удобен, т.к. позволяет создавать объектные библиотеки, которые потом можно использовать при программировании других задач, в том числе и на языках высокого уровня. Кроме того, появляется возможность коллективной разработки программы, когда каждый модуль разрабатывается отдельным программистом или группой программистов. В этой главе будет рассказано также о некоторых операторах макроассемблера и о работе со стандартным библиотекарем фирмы Микрософт.

I.

Рассмотрим вначале простой вариант - один основной модуль и один вспомогательный. Основным модулем мы будем называть тот, который получает управление после запуска программы. Два модуля представлены на Рис. 13.1 и Рис. 13.2 (основной, или главный, получает управление после запуска программы). Оттранслируем оба модуля с помощью MASM.EXE. В результате на диске образуются два объектных модуля: PR1.OBJ и PR2.OBJ, Теперь дело за редактором связей. В командной строке укажем: LINK PR2+PR1. После этого отвечайте на вопросы как обычно. При просьбе указать имя MAP-файла, укажите PR2 (впрочем, имя не имеет никакого значения). В результате компоновки образуется загрузочный модуль PR2.EXE и PR2.MAP. Содержимое MAP-файла показано на Рис. 13.3. В этом файле содержатся сведения о сегментах (начальный и конечный адрес, длина, имя, класс), скомпонованных в один модуль в том порядке, как они помещены в него (начиная с младших адресов). Обращаю Ваше внимание, что в начале идут сегменты модуля PR2, а затем сегменты модуля PR1. Именно так, как мы указали модули в командной строке. Если бы в командной строке было LINK PR1+PR2, то порядок сегментов был бы обратный. При этом сначала запустился бы модуль PR1, что привело бы к катастрофе. Итак, главный модуль должен быть первым, порядок остальных модулей уже не так важен (см., однако, ниже).

Представленные на Рис. 13.1 и 13.2 модули обладают следующей особенностью: из модуля PR2 вызывается процедура, находящаяся в модуле PR1, откуда, в свою очередь, вызывается процедура, находящаяся в модуле PR2. В конечном итоге происходит возврат в модуль PR2, и программа заканчивает свою работу.

В основе связки двух модулей лежат две команды транслятора: PUBLIC и EXTRN. Слово PUBLIC отмечает те имена (процедуры, метки), которые будут использоваться в других модулях. При определении имен с помощью слова PUBLIC тип имени не указывается, т.к. имя определено в самом модуле.

С помощью слова EXTRN отмечаются имена, определяемые в других **модулях**, - внешние имена. Определение внешних имен предполагает указание типа: для процедур и меток перехода указывается NEAR или FAR, для данных указывается BYTE, **WORD** или DWORD. Указание типа необходимо потому, **что** имя определено в другом модуле.

```
;модуль 1, программа PR1.ASM
DATA1 SEGMENT
TEXT DB 'Привет! Я в модуле 1. ',13,10,'$';
DATA1 ENDS
      PUBLIC WORK
      EXTRN PRI:FAR
LIB SEGMENT
      ASSUME CS:LIB
WORK PROC FAR
      PUSH AX
      PUSH DS
      MOV  AX,SEG TEXT
      MOV DS,AX
      MOV AH,9
      LEA DX,TEXT
      INT 21H
      POP DS
      POP AX
      CALL PRI  ;вызов процедуры, находящейся в модуле 2
      RETF
WORK ENDP
LIB ENDS
      END WORK
```

Рис. 13.1. Модуль 1 (программа PR1.ASM), компонуемая с PR2.ASM.

```
;модуль 2, программа PR2.ASM
DATA SEGMENT
TEXT1 DB 'Вызов процедуры PRI произошел из модуля 1.',13,10,'$'
DATA ENDS
SSEG SEGMENT STACK
      DB 40 DUP{?)
SSEG ENDS
      EXTRN WORK:FAR
      PUBLIC PRI
CODE SEGMENT
      ASSUME CS:CODE, DS:DATA, SS:SSEG
```

```
BEGIN:
      MOV  AX,DATA
      MOV  DS,AX
      CALL WORK      ;вызов процедуры, находящейся в модуле 1
      MOV AH,0
      INT 16H
      MOV AH,4CH
      INT 21H
;процедура будет вызвана из модуля 1
PRI PROC FAR
      PUSH AX
      PUSH DS
      PUSH DX
      MOV AH, 9
      LEA  DX,TEXT1
      INT  21H
      POP  DX
      POP  DS
      POP  AX
      RETF
PRI ENDP
CODE ENDS
      END BEGIN
```

Рис. 13.2. Модуль 2 (главный), компонуемый с модулем 1 (Рис. 13.1).

```
START   STOP    LENGTH  NAME  CLASS
00000H  0002BH  0002CH  DATA
00030H  00057H  00028H  SSEG
00060H  00080H  00021H  CODE
00090H  000A9H  0001AH  DATA1
000B0H  000C6H  00017H  LIB

PROGRAM  ENTRY  POINT  AT  0006:0000
```

Рис. 13.3. Содержимое файла PR2. MAP.

Указанный подход позволяет создавать программы, состоящие из отдельных модулей, разрабатываемых независимо друг от друга. Фактически появляется возможность коллективной разработки программ. Каждый разрабатывает свой модуль, удовлетворяющий определенным требованиям. В конце все модули объединяют при помощи редактора связей.

Вернемся, однако, к главе 11. В ней говорилось о способе нахождения конца программы путем включения в программу фиктивного сегмента. Этот сегмент должен распола-

гаться в конце (либо он последний в программе, либо имеет имя, которое после ассемблирования с опцией /А должно располагаться по алфавиту в конце ряда имен сегментов). По логике вещей понятно, что такой сегмент следует указать в модуле 1 (PR1.ASM). Проблема, однако, этим не решается - ведь модулей может быть несколько. Ну и что, скажете Вы, нужно компоновать так, чтобы модуль PR1.ASM был последним, в нем же последним должен идти фиктивный сегмент. Пусть это сегмент имеет имя ZSEG. Поставьте в этом модуле метку, скажем ZS, которую в начале программы определим как PUBLIC. В тех модулях, где мы собираемся использовать эту метку, точнее сегмент, где она находится - ведь смещение метки в этом сегменте есть просто О [34], необходимо отметить ее как внешнюю: EXTRN ZS:FAR. Наконец, чтобы узнать сегментный адрес этой метки (т.е. сегмент), можно выполнить команду: MOV AX,SEG ZS, и в AX будет помещен адрес сегмента, начинающегося за концом программы. И еще одна ситуация: Вы пишите лишь главный модуль, все остальные даны в виде объектных модулей (либо библиотек объектных модулей) - как быть в этом случае? Проблема решается элементарно - создайте еще один модуль (см. Рис. 13.4), а при компоновке поставьте его последним.

```
PUBLIC  ZS
ZSEG   SEGMENT
       ASSUME  CS:ZSEG
ZS:
ZSEG   ENDS
       END
```

Рис. 13.4. Модуль, используемый для определения конца программы.

Для упрощения работы с объектными модулями используется специальная программа - библиотекарь (LIB.EXE). В библиотеку объектных модулей можно поместить модули, которые содержат уже отлаженные процедуры. При компоновке следует на вопрос LIBRARIES [.LIB]: указать имя Вашей библиотеки. Если библиотек несколько, то их следует перечислить в строке, ставя между ними знак "+". Рассмотрим теперь основные команды библиотекаря:

LIB ИМЯ_БИБЛИОТЕКИ+ИМЯ_ОБЪЕКТНОГО_МОДУЛЯ - добавить к библиотеке новый объектный модуль. Если библиотеки с таким именем не существовало, то при выполнении данной команды она появится.

LIB ИМЯ_БИБЛИОТЕКИ-ИМЯ_ОБЪЕКТНОГО_МОДУЛЯ - удалить данный объектный модуль из библиотеки.

---

[34] Ситуация совсем не так проста. При определении сегмента можно указывать тип подгонки (PAGE - страница, PARA - параграф, WORD - слово, BYTE - байт). Согласно этому типу сегмент будет начинаться на границе страницы (100H байт), либо границе параграфа, либо на границе слова, либо выравнивания никакого не будет. По умолчанию всегда действует подгонка по началу параграфа. Если же взять тип подгонки сегмента, скажем, BYTE, смещение метки ZS в сегменте будет не нулевым, и это придется учитывать, в противном случае Вы рискуете испортить код программы.

LIBИМЯ_БИБЛИОТЕКИ*ИМЯ_ОБЪЕКТНОГО_МОДУЛЯ - после этой команды указанный объектный модуль образуется на диске с расширением OBJ. При этом он сохранится и в библиотеке.

LIB ИМЯ_БИБЛИОТЕКИ-*ИМЯ_ОБЪЕКТНОГО_МОДУЛЯ – делает то же, что и предыдущая команда, но с удалением объектного модуля из библиотеки.

LIBИМЯ_БИБЛИОТЕКИ,ИМЯ_ФАЙЛА - вывод каталога библиотеки в файл. Вместо имени файла можно использовать стандартные имена устройств: CON, PRN и т.д.

LIB ИМЯ_БИБЛИОТЕКИ-ИМЯ_1+ИМЯ_2 - удаляет из библиотеки объектный файл с именем ИМЯ_1 и заменяет его на объектный файл с именем ИМЯ_2. При этом требуется указать имя новой библиотеки.

П.

Ранее уже упоминалось о типах подгонки. Пора разобраться в этом подробнее, тем более что в книгах об этом не всегда четко говорится.

Когда упоминают, что сегмент может начинаться на границе слова, параграфа или начало его никак не выравнивается, то о чем собственно идет речь? Во многих книгах говорится, что сегмент начинается на границе параграфа. Возникает явное противоречие. Противоречие, однако, чисто методологического порядка. Нужно различать сегмент и то, что в нем находится. Сегмент всегда начинается с границы параграфа, тогда как для его содержимого это не всегда справедливо. Рассмотрим конкретный пример (Рис. 13.5). Вопрос: что будет содержаться в регистрах AX и BX после выполнения двух первых команд? Общий ответ гласит: в AX - адрес сегмента в параграфах (и это естественно, т.к. сегмент начинается на границе параграфа), в BX - смещение L1 в сегменте. Так как типом подгонки в этом случае является PARA, то данные в сегменте тоже должны начинаться на границе параграфа. Поэтому в BX должен содержаться 0. Теперь изменим тип подгонки на BYTE: DATA SEGMENT BYTE. Как и раньше, в AX будет содержаться адрес сегмента в параграфах. Но содержимое BX теперь уже не обязано быть равным 0.

Если в Вашей программе очень много сегментов, такая ситуация возникает часто при работе с языками высокого уровня, и есть смысл указывать у всех сегментов тип подгонки BYTE [35]. Это даст некоторое сокращение длины программы.

```
DATA  SEGMENT
L1    DB ?
        .
        .
DATA  ENDS
        .
        .
CODE  SEGMENT
        ASSUME DS:DATA, CS:CODE
```

[35] В языках высокого уровня это достигается путем установки определенной опции.

```
BEGIN:
        MOV AX,SEG L1
        MOV BX,OFFSET L1
        .
        .
CODE  ENDS
        END BEGIN
```

Рис. 13.5. Что будет содержаться в регистрах AX и BX?

III.

Вообще говоря, за директивой SEGMENT, кроме типа выравнивания, могут идти и другие операнды. Общий вид структуры сегмента следующий:

```
ИМЯ_СЕГМЕНТА SEGMENT [тип выравнивания] [тип объединения]
["класс"]
        .
        .
        .
ИМЯ_СЕГМЕНТА ENDS
```

Здесь в квадратные скобки взяты операнды, которые могут отсутствовать в определении сегмента. Класс представляет собой некоторое имя, взятое в кавычки. Тип объединения позволяет компоновщику объединять сегменты, имеющие одинаковые имена и одинаковые классы (классы могут отсутствовать). Перечислим типы объединения.

PUBLIC. При компоновке сегменты с таким типом объединения сцепляются в один так, что длина получившегося сегмента равна сумме длин отдельных сегментов. Порядок сцепления определяется порядком следования модулей в командной строке.

COMMON. Сегменты с таким типом объединяются в один сегмент с общим началом. Длина получившегося сегмента равна длине наибольшего сегмента.

STACK. Если среди сегментов имеется один сегмент с таким типом объединения, то это дает возможность компоновщику определить значения регистров SS и SP. SS устанавливается на начало сегмента, SP - на конец (старший адрес)[36]. Если имеется несколько сегментов с типом STACK, то они объединяются как по типу PUBLIC. Образуется один большой стек. SP будет указывать на дно этого стека, SS - на начало.

MEMORY. Данный тип объединения вызывает размещение сегмента в конец модуля (программы, если всего один модуль). Если связываются несколько сегментов с таким типом, то первый из них считается имеющим тип MEMORY, а остальные объединяются так, как тип COMMON.

---

[36] В главе 14 мы расскажем о структуре EXE-программ, в заголовке которых, содержится информация для загрузчика. Среди этой информации содержатся значения регистров CS, IP, SS,SP.

AT. После этого слова должен идти параграф - число или арифметическое выражение. С помощью данного типа можно задать определение данных и меток по фиксированным адресам памяти. Например, возможен следующий вариант сегмента:

```
VIDEO_BUF SEGMENT AT OB800H
L1  DB ?
VIDEO_BUF ENDS
```

Метка L1 фактически указывает на первый байт видеобуфера. Поэтому следующие команды приведут к тому, что на экране в левом верхнем углу появится значок "!":

```
.
MOV AX,SEG VIDEO_BUF
MOV ES,AX   MOV ES:L1,"!"
.
```

IV.

В этом разделе предлагается обсудить вопрос о передаче информации в процедуру и обратно. Вопрос не такой уж праздный, как может показаться, т.к. имеет прямой выход на стыковку ассемблера с языками высокого уровня.

Я знаю лишь три способа обмена информацией между процедурой и модулем, из которого эта процедура вызывается.

1. Обмен ведется через регистры. Это самый быстрый способ, хотя и не самый эффективный. Через регистры могут передаваться как сами данные, так и указатели на область памяти, где эти данные находятся. Например, указатель на строку, содержащую путь к файлу, принято передавать через пару регистров DS:DX.

Ограниченность этого метода заключается в том, что регистров не так уж много. Следовательно, через них нельзя передать большой объем информации. В принципе, однако, проблема разрешима. Можно поступить следующим образом: передать через регистры адрес области памяти (в языках высокого уровня это называют указателем), где будут располагаться данные. Структура этих данных, естественно, должна быть "известна" вызываемой процедуре.

2. Второй способ тоже достаточно очевиден. Предположим, в некотором сегменте имеется переменная типа BYTE: L1 DB ?. Посредством директив PUBLIC и EXTRN доступ к данной переменной может получить любая процедура, находящаяся в других модулях. Сегмент и смещение L1 получается обычным образом:

```
MOV AX, SEG L1
MOV ES,AX
MOV BX,OFFSET L1
```

Теперь ES:BX указывает на L1.

Если же процедура находится в том же модуле, откуда она вызывается, то в этом случае проблем еще меньше, т.к. она имеет доступ ко всем сегментам, находящимся в данном модуле.

3. Этот способ до некоторой степени является частным случаем первого. Однако в силу его специфики следует разобраться в нем подробно. Для простоты будем предполагать, что вызов процедуры будет длинным, т.е. в стек будет положено 4 байта. Изменения стека после вызова процедуры показано на Рис. 13.6. Команда RETF возвращает стек в исходное состояние.

Предположим, что мы хотим передать два параметра (типа WORD) в процедуру через стек. Поместим их соответственно в AX и в BX и выполним две команды: PUSH AX и PUSH BX. После этого выполним команду вызова процедуры. Изменения, происходящие со стеком в этом случае, отображены на Рис. 13.7. Теперь после входа в процедуру образовалась некоторая область, где содержатся входные параметры. Для доступа к ним удобно использовать регистр BP: MOV BP,SP, после этого BP+6 указывает на первый параметр (PUSH AX), а BP+4 — на второй параметр[37]. Для того чтобы удобнее было работать, можно сразу увеличить содержимое BP на 4: ADD BP,4. Теперь BP будет указывать на последний параметр.

Мы вплотную подошли к одному замечательному заключению. Стек можно использовать как сегмент для временного хранения данных для процедуры. Действительно, ведь перед вызовом процедуры можно зарезервировать некоторую часть стека для временного хранения данных во время работы процедуры. Для этого достаточно, перед тем как засылать параметры в стек уменьшить значение SP на нужное количество байт (размер области локальных переменных). Доступ к этой области также осуществляется через регистр BP. Возможны и другие варианты устройства области локальных переменных - например, изменение содержимого SP уже в самой процедуре. Область стека, где хранятся передаваемые параметры и локальные переменные, называется кадром. Пример с локальной переменной будет приведен ниже.

Следует иметь в виду, что часто значение регистра BP должно быть сохранено. Выполнение команды PUSH BP приведет к тому, что значение SP уменьшится на 2, поэтому на область параметров будет указывать BP+6.

При выходе из процедуры состояние стека должно быть восстановлено. Если нет параметров, то состояние стека восстанавливается командой RETF. Если какая-то часть стека была зарезервирована, как это показано на Рис. 13.7, то командой восстановления будет RETF N, где N - число байт, на которые нужно сдвинуть SP (в сторону старших адресов) после извлечения адреса возврата.

В стеке можно передавать как сами параметры, так и их адреса. В случае, если нужно передать целый массив данных, удобнее, чтобы не тратить зря область стека, передавать не сам массив, а адрес первого элемента (4 или 2 байта).

Если Вы хотите, чтобы параметры передавались и из процедуры, то состояние стека восстанавливать не надо. В этом случае SP по выходу из процедуры будет показывать на начало области параметров. Возвратить же стек в исходное состояние можно командой SUB SP,N, где N - размер области параметров.

Передача параметров посредством стека взята на вооружение языками высокого уровня. Мы воспользуемся материалом главы, посвященной языкам высокого уровня (см. главу 15 и особенно главы 24, 25).

---

[37] Напомню читателю, что регистр BP по умолчанию показывает смещение в сегменте стека, т.е. команды MOV AX,[BP+2] и MOV AX,SS:[BP+2] идентичны (см. главу 4.).

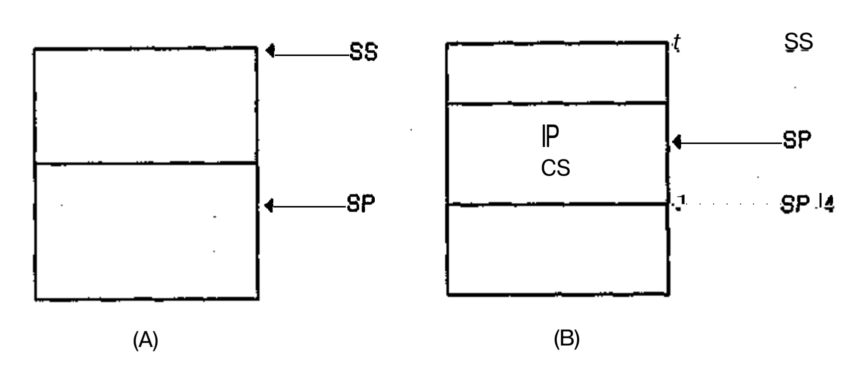


Рис. 13.6. **Стек** *до (A) и после (B) вызова процедуры. Пунктиром отмечено старое положение указателя стека.*

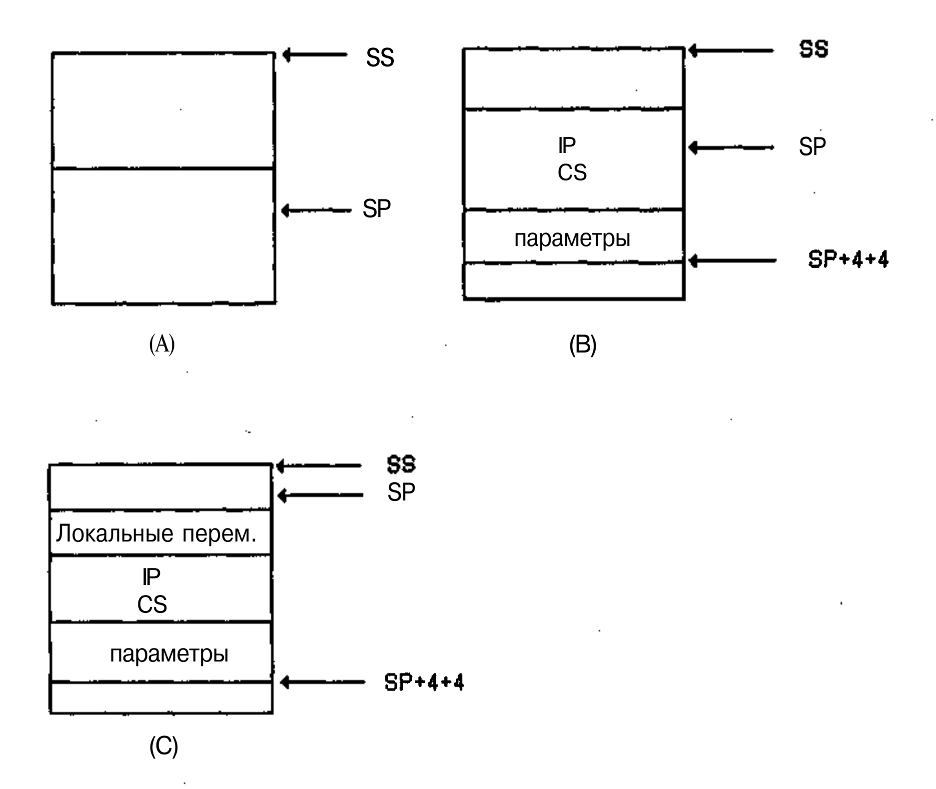


Рис. 13.7. **Стек** *до (A) и после (B) вызова процедуры.* **(C)** *локальные переменные. Перед вызовом процедуры выполнены команды PUSH* ***AX*** *и* ***PUSH BX.***

Если в процедуре необходимо использовать локальные переменные, то место в памяти для них резервируется следующим образом. Содержимое SP увеличивается на общую длину для всех локальных переменных. Память между старым и новым значениями SP и будет областью для локальных переменных (Рис. 13.7(С)).

Чтобы не быть голословным приведу следующую программу. Можно считать ее некоторым итогом главы. Данная программа будет состоять из двух модулей. Во втором модуле будут находиться процедуры, которые получают параметры через стек. Кроме того, в одной из процедур мы определим локальную переменную и будем ее использовать. Вызов и работа процедур будет весьма напоминать то, как это делается в языках высокого уровня. Обращу Ваше внимание еще и на то, что процедуры вполне рекурсивны, т.е. могут вызываться сами из себя, как это происходит в таких языках высокого уровня, как Паскаль и Си.

```
;программа loc.asm
;имена процедур, используемых из других модулей
EXTRN COPY:FAR
EXTRN PRINT:FAR
;сегмент стека
STE SEGMENT STACK
      DB 100 DUP(0)
STE ENDS
;сегмент кода
CODE SEGMENT
      ASSUME CS:CODE, DS:DATA, SS:STE
BEGIN:
;начальные установки
      MOV AX,DATA
      MOV DS,AX
;вызов процедуры
;прототип процедуры INT COPY(BUFER,STR1,STR2)
;BUFER, STR1, STR2 - указатели на строки
;длинный указатель представляет собой совокупность
;сегментного адреса и смещения
;в результате выполнения строка STR2 добавляется к строке
;STR1 и результат помещается в BUFER
;конец строк отмечается кодом 0
;функция возвращает длину получившейся строки
;возврат, как и положено, осуществляется через AX
;указатели на строки будем помещать в стек слева направо
      PUSH DS
      LEA AX,BUFER
      PUSH AX
      PUSH DS
      LEA AX,STROKA1
      PUSH AX
```

```
       PUSH DS .
       LEA  AX,STROKA2
       PUSH AX
       CALL COPY
;вывод содержимого буфера
;прототип процедуры VOID PRINT(BUFER)
;процедура выводит содержимое буфера, в конце буфера должен
;стоять код 0
       PUSH DS
       LEA  AX,BUFER
       PUSH AX
       CALL PRINT
;конец программы
       MOV  AX,4C00H
       INT  21H
CODE ENDS
;сегмент данных
DATA  SEGMENT
STROKA1  DB  "Первая строка"  ,0
STROKA2  DB  "Вторая строка",0
BUFER    DB  100 DUP(?)
DATA ENDS
       END  BEGIN
```

Рис. 13.8. Основная программа конкатенации двух строк.

```
;программа locl.asm
/имена внешних процедур
PUBLIC COPY, PRINT
;сегмент кода
COD SEGMENT
       ASSUME CS:COD
/процедура копирования двух строк в буфер
COPY PROC
       PUSH BP
       MOV  BP,SP
       SUB  SP,2  /локальная переменная [BP-2]  (тип WORD)
/параметры [BP+6] - строка STROKA2
;[BP+10] - строка STROKA1,  [BP+14] - буфер
       MOV  WORD PTR [BP-2],0 /обнулить локальную переменную
       LDS  SI,[BP+10]          ;STROKA1
       LDS  DI,[BP+14]          ;буфер
LO1:
       MOV  AL,DS:[SI]
       CMP  AL,0
       JZ   ZER1
       MOV  DS:[DI],AL
```

```
        INC   DI
        INC   SI
        INC   WORD  PTR  [BP-2]       ;счетчик
        JMP  SHORT  L01
ZER1:
        LDS  SI,[BP+6]               ;STROKA2
LO2:
        MOV   AL,DS:[SI]
        CMP   AL,0
        JZ    ZER2
        MOV   DS:[DI],AL
        INC   DI
        INC   SI
        INC   WORD  PTR  [BP-2]       ;счетчик
        JMP   SHORT  LO2
ZER2:
        MOV   BYTE  PTR  DS:[DI],0  ;0 - завершение строки
        MOV   AX,[BP-2]               ;в AX - длина строки в буфере
;освободить память, которую занимала локальная переменная
        ADD   SP,2
        POP   BP
;выход с освобождением стека
        RETF  16
COPY  ENDP
;вывод строки, адрес строки передается через стек
;строка должна заканчиваться кодом О
PRINT  PROC
        PUSH  BP
        MOV   BP,SP
        LDS   SI,[BP+6]             ;BUFER
LO3:
        MOV   DL,DS: [SI]
        CMP   DL,0
        JZ    ZER3
        MOV  AH, 2
        INT   21H
        INC   SI
        JMP    SHORT  LO3
ZER3:
        POP   BP
        RETF  4
PRINT  ENDP
COD ENDS
        END
```

Рис. 13.9. Модуль для программы конкатенации.

Разберите программу на рисунках 13.8 и 13.9. Она написана в стиле языка высокого уровня с передачей параметров через стек и использованием локальных переменных. Обратите внимание, что процедура (в терминах языка высокого уровня это функция) COPY возвращает длину получившейся строки. Видоизмените программу так, чтобы процедура PRINT требовала еще и длину выводимой строки, но заканчивала вывод, если встречается код 0. Если Вам пока еще не все понятно, вернитесь к этой программе после главы 15.

Глава 14. Структура информации на диске.

> ***- Нет, виноват! Разоблачение совершенно необходимо. Без этого ваши блестящие номера оставят тягостное впечатление.***
>
> *М.А. Булгаков.*
> *Мастер и Маргарита.*

Материал настоящей главы носит справочный характер (см. [5,9,11,17]). Для того чтобы досконально разобраться в излагаемом здесь материале, вооружитесь каким-нибудь дисковым редактором (например, DISKEDIT из пакета Norton Utilites) и тщательно проверьте все, о чем мы будем Вам рассказывать. Здесь приводятся данные как для операционной системы MS DOS, так и для Windows. Однако справочная информация по Windows нами ограничена вследствие ограниченности объема книги. Всем желающим разобраться в структуре информации для Windows, рекомендуются книги [27,29].

I. Структура EXE-программ для MS DOS.

Смещение	Длина	Имя	Комментарий
+0	2	MZ	подпись, признак EXE-программы
+2	2	PartPag	длина неполной последней страницы
+4	2	PageCnt	длина в страницах (512 б.), включая заголовок и последнюю страницу
+6	2	ReloCnt	число элементов в таблице перемещения
+8	2	HdrSize	длина заголовка в параграфах
+0AH	2	MinMem	минимум требуемой памяти за концом программы
+0CH	2	MaxMem	максимум требуемой памяти за концом программы
+0EH	2	ReloSS	сегментный адрес стека
+10H	2	EXESp	значение регистра SP
+12H	2	ChkSum	контрольная сумма
+14H	2	ExeIP	значение регистра IP
+16H	2	ReloCS	сегментный адрес кодового сегмента
+18H	2	TablOff	смещение в файле первого элемента таблицы перемещения
+1AH	2	Overlay	номер оверлея, 0 для главного модуля
* Конец форматированной порции заголовка **			
+1CH			
** Начало таблицы перемещения (возможно с 1CH) **			
+?	4*?	смещ. сегмент ... смещ. сегмент	

Рис. 14.1. Структура EXE-заголовка.

```
DSEG   SEGMENT
TEXT DB 'Обычная EXE-программа.','$',13,10
DSEG   ENDS
SSEG SEGMENT  STACK
      STAC DW 60 DUP(?)
SSEG ENDS
CODSEG   SEGMENT
      ASSUME  CS:CODSEG, DS:DSEG, SS:SSEG
BEGIN:
      MOV AX,SEG TEXT    ;*
      MOV DS,AX
      MOV AX,SEG STAC    ;*
      MOV SS,AX
      LEA DX,TEXT
      MOV AH,9
      INT 21H
      MOV AH,4CH
      INT 21H
CODSEG  ENDS
      END BEGIN
```

Рис. 14.2. Текст простой программы.

Заголовок EXE-программы.

```
0000: 4D 5A B6 00 02 00 02 00 - 20 00 00 00 FF FF 02 00 MZ..............
0010: 78 00 16 FD 00 00 OA 00 - 1E 00 00 00 01 00 01 00 x...............
0020: OA 00 06 00 OA 00 00 00 - 00 00 00 00 00 00 00 00 ................
0030: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0040: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0050: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0060: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0070: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0080: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0090: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
OOAO: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
00B0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
OOCO: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
OODO: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
OOEO: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
OOFO: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0100: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0110: 00 00 00 00 00 00 00 00 - 0.0 00 00 00 00 00 00 00 ................
0120: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0130: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0140: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
```

```
0150: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0160: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0170: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0180: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0190: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
01A0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
01B0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
01C0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
01D0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
01E0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
01F0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
```

Начало образа задачи.

Сегмент данных (сегментный адрес 00H):

```
0200: 8E A1 EB E7 AD A0 EF 20 - 45 58 45 2D AF E0 AE A3 Обычная EXE-прог
0210: E0 A0 AC AC A0 2E 24 0D - 0A 00 00 00 00 00 00 00 рамма.$........
```

Сегмент стека (сегментный адрес 02H):

```
0220: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0230: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0240: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0250: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0260: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0270: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0280: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
0290: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ................
```

Сегмент кода (сегментный адрес 0AH):

```
02A0: B8 00 00 8E D8 B8 02 00 - 8E D0 8D 16 00 00 B4 09 ................
02B0: CD 21 B4 4C CD 21 00 00 - 00 00 00 00 00 00 00 00 .!.L.!..........
```

Рис. 14.3. Код EXE-программы (Рис. 14.2) на диске.

На Рис. 14.1 представлена структура заголовка EXE-программ. Необходимость наличия такого заголовка заключается в том, что EXE-программа состоит в общем случае из нескольких сегментов. Это, в свою очередь, приводит к появлению команд, в которых явно фигурируют адреса сегментов (длинные JMP или CALL, команды типа MOV AX,SEG метка). С другой стороны - программа может быть помещена в любую область памяти. Поэтому эти адреса требуют неизбежной корректировки. Этим и занимается загрузчик, когда запускает EXE-программу, а нужную информацию он получает из заголовка.

Замечу, что структура ЕХЕ-программ довольно гибкая. Так, длина ЕХЕ-модуля может быть длиннее загружаемой части, В оставшейся части могут находиться данные, оверлеи и т.п.

Корректировка адресов происходит по таблице перемещения, где содержатся адреса всех слов, представляющих собой сегментные адреса. На Рис. 14.2 представлена простая программа, дисковая копия ЕХЕ-модуля которой показана на Рис. 14.3. Пользуясь таблицей на Рис. 14.1, находим, что таблица перемещения начинается со смещения 1ЕН. Длина ее равна 2 элементам (по 4 байта каждый):

1-й элемент 01 00 OA 00, т.е. OOOAH:0001H - адрес в первой команде; 2-й элемент 06 00 02 00, т.е. 0002H:0006H - адрес в третьей команде. В других командах сегментные адреса не используются, поэтому таблица перемещения состоит всего из двух элементов.

Ниже таблицы перемещения находится непосредственно образ задачи. Первому сегменту, находящемуся в образе, присваивается сегментный адрес 0. Замечу, что единственный сегмент, положение и размер которого можно вычислить из заголовка, - это сегмент стека. Для сегмента кода можно получить только начало. Всю остальную информацию можно узнать лишь путем анализа кода программы.

Расшифруем сегмент кода:

```
B8 00 00     MOV AX,0        ; 0- адрес сегмента данных
8E D8        MOV DS,AX
B8 02 00     MOV AX,02       ; 2- адрес сегмента стека
8E DO        MOV SS,AX
8D 16 00 00  LEA DX,[0000]   ; 0- смещение строки в сегменте стека
B4 09        MOV AH,09
CD 21        INT 21H
B4 4C        MOV AH,4CH
CD 21        INT 21H
```

Обращу Ваше внимание еще на один важный момент. Количество памяти в байтах, необходимое для загрузки ЕХЕ-программы в ОЗУ, вычисляется по формуле (PageCnt-1)*512+PartPag. Довольно часто это число совпадает с длиной программного модуля. Если же длина модуля оказывается длиннее, то это может быть по двум причинам (исключая вирус):

а) модуль удлинился вследствие сбоя ОС,

б) за загружаемой частью программа хранит оверлей или данные.

ЕХЕ-заголовки для программ работающих под Windows (16-битный вариант).

16-битные программы, работающие в среде Windows 95 (а также OS/2 версии 1), имеют следующую структуру:

заголовок DOS-программы
DOS-программа, выводящая простое сообщение о невозможности работать в среде MS DOS
Windows-заголовок
Windows-программа

Легко понять, что при запуске программы в среде DOS запускается DOS-программа, которая после вывода сообщения возвращает управление операционной системе. При запуске же программы в среде Windows загрузчик распознает, что данная программа есть приложение Windows (в заголовке имеется сигнатура PE или NE) и запускает ее как положено. Ниже приводится структура заголовка для 16-битных приложений под Windows.

Смещение	Длина	Содержание
+0	2	NE - подпись, признак EXE-программы для Windows
+2	1	номер версии компоновщика
+3	1	номер модификации компоновщика
+4	2	смещение таблицы элементов
+6	2	число байтов в таблице элементов
+8	4	32-битовый CRC всего файла
+CH	2	ключевое слово: 0000h - NOAUTODATA 0001h - SINGLEDATA 0002h - MULTIPLEDATA 2000h - ошибка во время редактирования 8000h - библиотечный модуль
+EH	2	номер сегмента с данными типа automatic
+10H	2	начальный размер динамической области, добавленной к DS
+12H	2	начальный размер стека, добавленный к DS (0 - DS не равно SS)
+14H	4	CS:IP
+18H	4	SS:SP
+1CH	2	число элементов в таблице сегментов
+1EH	2	число байтов в таблице с не резидентным именем
+20H	2	смещение таблицы сегментов (относительно начала этой секции заголовка)
+22H	2	смещение таблицы ресурсов (...)
+24H	2	смещение таблицы с резидентным именем (...)
+26H	2	смещение таблицы ссылок модуля (...)

+28H	2	смещение таблицы внешних имен (...)
+2AH	4	смещение таблицы с нерезидентным именем (относительно начала файла)
2EH	2	число точек перемещаемых элементов
30H	2	счетчик сдвига при выравнивании логического сектора (логарифм по основанию 2, от размера сектора сегмента)
32H	2	число резервных сегментов
34H	10	зарезервировано

EXE-заголовки для программ, работающих под Windows (32-битный вариант).

Не буду утомлять внимание читателей описанием полей PE-заголовка. Все интересующиеся смогут найти эту структуру, обратившись к заголовочному файлу WINNT.H из пакета C++ для Windows. Эта структура называется IMAGE_NT_HEADERS.

II. Средство работы с диском.

Все возможности операционной системы MS DOS для работы с дискетами и жесткими дисками основываются на прерывании 13H.

Прерывание 13H дает весьма обширный сервис работы с дисками на физическом уровне. Среди функций, выполняемых данным прерыванием, - дать статус ошибки последней дисковой операции, проверить сектор, форматировать дорожку, дать параметры диска и т.д. Мы разберем лишь чтение-запись с помощью данного прерывания, а также форматирование дорожки диска. Подробнее о нем можно прочесть, например, в [13,17] или в нашем справочнике (см. Приложение 8).

Чтение. -
Вход:
AH - 02,
DL - номер диска (0 - A, 1 - B; для твердого 80H, для второго-81H)
DH - номер головки,
AL - число читаемых секторов,
ES:BX - буфер для чтения,
CX - номер сектора и цилиндра: первые 6 бит - номер сектора, остальные - номер цилиндра, причем 8 и 7 бит следует поставить перед другими (8-10,7-9).
0:0078H - адрес таблицы параметров дискеты (Рис. 14.10).
Выход:
если взведен флаг C, то код ошибки в AH.
Запись. Аналогична чтению.
Форматирование дорожки.
Вход:
AH - 05,

AL - число секторов на дорожке или коэффициент чередования в случае жесткого диска,

CX - номер сектора и цилиндра (так же, как для чтения-записи),

DL - номер устройства,

DH - номер головки,

ES:BX - буфер с информацией для разметки. Для дискеты это последовательность четырехбайтных величин: 1-й - номер дорожки, 2-й - номер головки, 3-й - номер сектора, 4-й - длина сектора (0 - 128, 1 - 256, 2 - 512, 3 - 1024). Для жесткого диска буфер состоит из пар байт: 1-й либо 0 -разметить как обычный сектор, либо 80H - разметить как дефектный сектор, 2-й номер сектора. Для некоторых моделей контролеров жестких дисков содержимое буфера игнорируется.

Меняя содержимое буфера, можно произвести нестандартное форматирование. Например, изменить нумерацию секторов на дорожке. В некоторых случаях изменения буфера недостаточно - требуется изменить содержимое таблицы параметров дискеты или жесткого диска. Такая ситуация может возникнуть при форматировании с меньшим числом секторов, чем предусматривает дисковод. Если дорожка отформатирована нестандартно, то для чтения-записи на нее также необходимо менять таблицу параметров.

Выход: AH - код ошибки, если флаг C установлен. Возможна ситуация, когда BIOS не поддерживает форматирование жесткого диска. В этом случае в AH будет возвращена ошибка 1 - нераспознанная команда, и Вам не останется ничего иного, как форматировать посредством обращения к контролеру жесткого диска.

Прерывания 25H, 26H. Данные прерывания DOS осуществляют чтение-запись логических секторов (в каждом разделе или гибком диске сектора пронумерованы от 0 до n-1, п - число секторов в разделе или гибком диске).

Чтение. Вход: AH - 25H, AL - номер устройства (0 - A, 1 - B, 2 - C и т.д.), CX - количество секторов,

DX - номер сектора,

DS:BX - адрес буфера.

Выход:

если взведен флаг C, то код ошибки в AX.

В стеке остается лишнее слово (см. главу 12, перехват 25/26 прерываний).

Запись. Аналогична чтению. Начиная с DOS 4.0 появилась новая возможность работы с устройствами емкости свыше 32 Мб.

Если в CX FFFFH, то:

DS:[BX] - номер сектора (DWORD),

DS:[BX+4] - количество секторов (WORD),

DS:[BX+6] - смещение буфера (WORD),

DS:[BX+8] - сегмент буфера (WORD).

Здесь может возникнуть следующая проблема. Старые ОС не поддерживают последнее. Поэтому в своей программе нужно предусмотреть возможность запуска в среде ОС более ранних версий. Можно поступить следующим образом: попробовать вначале осуществить операцию чтения/записи новым методом; если будет установлен флаг ошибки, попытаться осуществить действие старым способом.

III. Сектора загрузки.

Начнем с первого сектора дискеты. Логический номер этого сектора 0, а физические координаты — (1,0,0)[38]. Это BOOT-сектор. В нем содержатся данные о дискете (Рис. 14.4), а также программа для запуска системных файлов. Формирование BOOT-сектора происходит во время форматирования дискеты.

Обратимся теперь к жесткому диску. Каждый раздел жесткого диска также начинается с BOOT-сектора, логический номер которого 0. У жесткого диска имеется и Главная загрузочная запись (Master Boot Sector - MBS). Физические координаты ее такие же, как у BOOT-сектора дискеты. Как и BOOT-сектор она состоит из двух частей: вначале идет небольшая программа, а затем таблица разделов (Partition Table) (Рис. 14.5). В таблице разделов содержатся сведения о том, какие разделы имеются на жестком диске, типы этих разделов и их физические координаты (Рис. 14.6). При загрузке системы с винчестера BIOS считывает MBS по абсолютному адресу 0:7C00H и передает управление программе. Программа, руководствуясь Partition Table, находит системный раздел и загружает в память BOOT-сектор этого раздела, после чего передает ему управление. Далее все идет так же, как в случае с дискетой.

Вернемся снова к MBS. Изучая его для винчестеров с разным количеством разделов, Вы обнаружите, что число разделов, которые указываются в Partition Table, никогда не превышает 2. Второй раздел называется расширенным (Extended). Оказывается, он имеет свой MBS, только без загрузочной программы. В его Partition Table также имеются ссылки на один или два раздела, один из которых имеет тип Extended и т.д. Изложенное схематично изображено на Рис. 14.7. Таким образом, имеется возможность работать с любым количеством логических разделов на жестком диске любого объема. Такая структура в программировании называется связанным списком.

Смещение	Размер байт	Содержание
00H	3	Короткий или длинный JMP на программу загрузки
03H	8	Имя и версия
0BH	2	Количество байт на сектор
0DH	1	Количество секторов на кластер
0EH	2	Количество резервных секторов, включая BOOT-сектор
ЮН	1	Число таблиц FAT
11H	2	Максимальное количество элементов в корневом оглавлении
13H	2	Общее количество секторов на логическом диске (<32 Мб)
15H	1	Тип носителя
16H	2	Количество секторов в одной FAT
18H	2	Количество секторов на трек
1AH	2	Количество головок

[38] На физическом уровне подсчет секторов проводится в следующем порядке: вначале сектор, затем головка, затем цилиндр.

1CH	4	Количество "скрытых" секторов
20H	4	Общее количество секторов на логическом диске
24H	1	Физический номер диска
25H	1	Зарезервировано
26H	1	Сигнатура 29H
27H	4	Двоичный номер диска
28H	11	Метка диска
36H	8	Зарезервировано
3EH	Программа загрузки	
1FEH	2	Сигнатура OAA55H

Рис. 14.4. Boot-сектор. Начиная со смещения ВН и заканчивая смещением 16Н, расположен BPB - BIOS Parameters Block.

Байты	Размер в байтах	Содержание	
00H-1BDH	446	Программа главной загрузки	
1BEH-1CDH	16	Элемент первого раздела	Partition Table
1CEH-1DDH	16	Элемент второго оглавления	
1DEH-1EDH	16	Элемент третьего оглавления	
1EEH-1FDH	16	Элемент четвертого оглавления	
1FEH-1FFH	2	Сигнатура OAA55H	

Рис. 14.5. Главная загрузочная запись (MasterBoot-сектор).

Смещение	Размер байт	Содержание
00H	1	Признак загрузки 80H - загружаемый раздел OOH - незагружаемый раздел
		Начало раздела диска
01H	1	бит 0-7:номер головки (0-255)
02H	1	бит 0-5:номер сектора (1-63) бит 6-7:старшие биты ном.цилин.
03H	1	бит 0-7: младшие биты ном.цилин. (0-1023)
		Тип раздела
04H	1	00 - раздел не используется 01H - DOS 2.X с 12-бит. FAT 04H - DOS 3.X с 16-бит. FAT

		05H - DOS 3.3 Extended-раздел 06H - DOS 4.X с 16-бит. FAT
		Конец раздела диска
05H	1	бит 0-7:номер головки (0-255)
06H	1	бит 0-5:номер сектора (1-63) бит 6-7:старшие биты ном.цилин.
07H	1	бит 0-7:младшие биты ном.цилин. (0-1023)
		Относительный сектор
08H	4	Количество секторов перед началом раздела
		Раздел
0CH	4	Количество секторов в разделе

Рис. 14.6. Формат полей описания раздела диска.

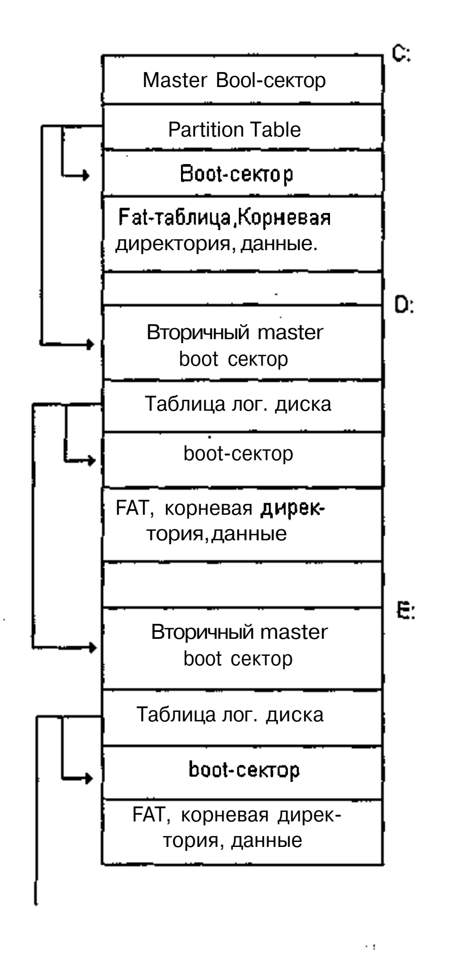


Рис. 14.7. Общая структура логических дисков винчестера.

Ниже (Рис. 14.8) приводится простой пример работы с корневыми секторами жесткого диска. Программа читает название фирмы-изготовителя программы, с помощью которой был отформатирован первый раздел жесткого диска.

```
DATA SEGMENT
BUFER DB 512 DUP(?)
DATA ENDS
STT SEGMENT STACK
      DW 40 DUP(?)
STT ENDS
CODE SEGMENT
      ASSUME CS:CODE, DS:DATA, SS:STT
BEGIN:
;установка сегментных секторов
      MOV AX,DATA
      MOV DS,AX
      MOV ES,AX
      MOV AX,STT
      MOV SS,AX
;читать главную загрузочную запись винчестера
      LEA BX,DS:BUFER
      MOV AX,0201H
      MOV DX,0080H
      MOV CX,0001H
      INT 13H
;определяем положение загрузочного сектора
;первого раздела жесткого диска
      MOVDH,DS:BUFER[1BFH]             ;1BEH+1
      MOV CX,WORD PTR DS:BUFER [1COH] ;1BEH+2
      MOV AX,0201H
;читаем загрузочный сектор
      INT 13H
      MOV BYTE PTR DS:BUFER [3+8],'$'   ;отмечаем конец строки
;выводим создателя загрузочного сектора
      MOV DX,BX
      ADD DX,3
      MOV AH,9
      INT 21H
KON:
      MOV AH,4CH
      INT 21H
CODE ENDS
      . END BEGIN
```

Рис. 14.8. Программа чтения имени и версии из первого раздела жесткого диска.

Простая программа на **Рис. 14.9** читает главную загрузочную запись и проверяет правильность сигнатуры. Если сигнатура не верна, то программа автоматически исправляет ее.

```
CODE  SEGMENT
      ASSUME CS:CODE, DS:CODE, SS:CODE, ES:CODE
      ORG  100H
BEGIN:
;чтение главной загрузочной записи первого жесткого диска
      MOV  DL,80H  ;первый жесткий диск
      MOV  AH,2    ;чтение
      MOV  DH, 0   ;номер головки
      MOV  AL,1    ;читаем один сектор
      MOV  CX, 1   ;номер сектора 1, номер цилиндра 0
      LEA  BX,BUF  ;ES:BX на буфер
      PUSH CS
      POP  ES
      INT  13H
      JC   NO_R
;проверка сигнатуры
      CMP  WORD PTR BUF+1FEH, 0AA55H
      JZ   OK
;исправим сигнатуру
      LEA  DX,MES4
      MOV  AH, 9
      INT  21H
      MOV  WORD PTR BUF+1FEH,0AA55H
      MOV  AH,3    ;запись
      MOV  DL,80H
      MOV  DH,0
      MOV  AL,1
      MOV  CX,1
      LEA  BX,BUF
      INT  13H
      JC   NO_W
      RET
OK:
      LEA  DX,MES3
      MOV  AH, 9
      INT  21H
      RET
NO_R:
      LEA  DX,MES1
      MOV  AH, 9
```

```
        INT 21H
        RET
NO_W:
        LEA DX,MES2
        MOV AH,9
        INT 21H
        RET
BUF   DB 512 DUP(?)
MES1  DB 'Ошибка чтения',13,10, '$'
MES2  DB 'Ошибка записи',13,10, '$'
MES3  DB 'Сигнатура Ок!',13,10, '$'
MES4  DB 'Сигнатура отсутствует,  исправляем.',13,10,'$'
CODE  ENDS
        END BEGIN
```

Рис. 14.9. Проверка и исправление сигнатуры жесткого диска.

IV. Расположение файлов на диске.

Прежде введем следующие обозначения: количество байт на сектор - байт_на_сектор, количество секторов на кластер - сек_на_кл, количество резервных секторов, начиная с BOOT-сектора - рез_сек., число FAT-таблиц - кол_FAT, максимальное количество элементов в корневом каталоге - эл._кат., количество секторов в одной FAT-таблице - сек._FAT. Все эти данные хранятся в BOOT-секторе.

Расположение файла на диске определяют две структуры: FAT-таблица [39] и каталоги [40]. Различают корневой [41] каталог и подкаталоги. Подкаталоги фактически представляют собой обычные файлы, имеющие специальный атрибут. Корневой же каталог имеет фиксированную длину и место положения. Длина одного элемента каталога равна 32 байтам. Отсюда длина каталога в килобайтах составляет:

32*эл._кат./1024, при эл._кат.=512 получим 16К, или 32 сектора. Начало корневого каталога (логический сектор): рез._сек.+сек._FAT*кол._FAT.

Начало области данных (логический сектор):

рез._сек.+сек._FAT*кол._FAT+32*эл._кат./байт_на_сектор. Элемент каталога имеет структуру, представленную на Рис. 14.10.

В элементе каталога содержится номер начального кластера. Кластер является той единицей деления файла, которой непосредственно оперирует операционная система. Расположение кластеров файла на логическом диске содержится в FAT-таблице. К изложению структуры FAT-таблицы мы сейчас приступаем.

---

[39] FAT - file allocation table, т.е. таблица размещения файлов.

[40] Запись FAT-таблица, как Вы, наверное, понимаете, тавтология. Мы, однако, пользуемся этим термином, сознавая некоторую некорректность.

[41] Название "корневой" довольно уместно, если представить систему каталогов в виде дерева.

Смещение	Длина	Содержимое
0	8	Имяфайла
8	3	Расширение
0BH	1	Атрибут файла
0CH	0AH	Резерв
16H	2	Время создания или последней модификации
18H	2	Дата создания или последней модификации
1AH	2	Номер начального кластера
1CH	4	Размер файла

Рис. 14.10. Структура элемента каталога.

FAT-таблица состоит из элементов. Элемент может быть как 12-битным, так и 16-битным. Тип элемента указан в Partition Table. Каждый элемент соответствует некоторому кластеру на диске (по порядку). Если кластер принадлежит файлу, то в соответствующем ему элементе FAT-таблицы содержится ссылка на следующий кластер. Таким образом, зная начало файла из каталога, по FAT-таблице можно отыскать весь файл. Надо только иметь в виду, что ссылки начинаются с третьего элемента. Первым байтом в таблице является дескриптор. В ранних версиях MS DOS он однозначно определял тип носителя (формат дискеты, раздел винчестера, электронный диск). Следующие семь байт (5 байт в 12-битной таблице) содержат FFH. В FAT-таблице приняты следующие обозначения:

```
(0)000H               кластер доступен,
(F)FF0H-(F)FF6H       зарезервированный кластер,
(F)FF7H               плохой кластер,
(F)FF8H-(F)FFFH       конец цепочки,
(0)002H-(F)FEFH       номер следующего кластера.
```

Старшая цифра в скобках относится к 16-битной FAT.

Приведу полезную формулу получения номера логического сектора из номера кластера: обл._дан.+(кластер-2)*сек._на_кл., здесь обл._дан. - начало области данных.

Как уже отмечалось, для того чтобы определить, какая имеется FAT-таблица, необходимо обратиться к таблице разделов. Таблица разделов при этом может ссылаться на другую таблицу, а та, в свою очередь, на третью и т.д. Поэтому получить параметры данной FAT-таблицы не всегда простая задача. Есть, однако, более простые способы определения типа FAT-таблицы. Пользуйтесь таким правилом: если общее количество секторов в разделе больше 20740, то используется 16-битная FAT-таблица, в противном случае используется 12-битная таблица. Общее количество секторов можно определить из BOOT-сектора.

Ниже приводится значение битов атрибута файла для операционной системы MS DOS. Как уже было сказано, под атрибут отводится один байт.

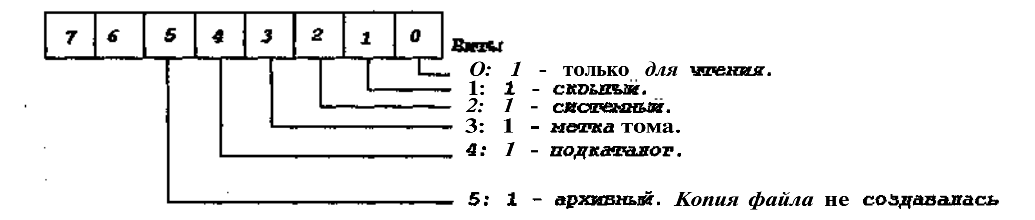


Рис. 14.11. Атрибут файла.

При работе с атрибутами помните, что поиск по атрибуту функций FindFirst и FindNext включающий. Т.е. ищутся файлы, имеющие данный бит. Если Вы, например, ищете все директории, то следует искать файлы со всеми атрибутами: директорий, архив, скрытый и только для чтения. Затем уже по атрибуту найденного файла можно определить, является он каталогом или нет. Надо иметь в виду, что стандартными методами DOS можно менять биты 0, 1, 2, 5. Биты третий и четвертый можно изменить, только обратившись к диску в обход стандартных функций MS DOS.

В операционной системе Windows под атрибут файла отводится четыре байта, т.е. 32 бита. Однако количество определенных битов стало не намного больше. К выше определенным битам добавились также: Бит 7 - нормальный (нет других атрибутов), Бит 8 - временный, Бит 10 - сжатый, Бит 11 – офлайновый (данные файла могут быть недоступны в настоящее время). Для работы с атрибутами файл в Windows имеются две API функции GetFileAttributes и SetFileAttributes.

Файловое время.

В операционной системе каждому файлу ставится в соответствие время и дата создания. При этом при изменении файла время и дата создания заменяются на время и дату модификации. Под время и дату отводится по слову. Слово времени: часы - 5 старших бит, минуты - 6 бит (с 5 по 10), секунды - 5 первых бит. Причем хранится количество секунд, деленное на 2 (иначе не хватит места). Получить или изменить дату или время файла можно посредством функции 57Н (см. Приложение 6).

Иная ситуация в операционной системе Windows. Каждому файлу ставится три времени (время и дата): время создания, время последней модификации, время последнего доступа. Причем время хранится в 100-наносекундных интервалах как промежуток времени, начиная с 1600 года. На это отводится три 64-битных слова. Устанавливать и получать файловое время можно при помощи API-функций GetFileTime и SetFileTime. Заметим, кстати, что в Windows появилась возможность сортировать файлы на файлы, часто (редко) модифицируемые, и файлы с частым (редким) доступом, что можно использовать для оптимизации файловой системы.

В конце раздела приводятся структуры данных для гибкого и жесткого дисков, определяющие параметры, необходимые для основных операций с дисками.

Смещение	Длина	Содержание
40	1	биты 0-3: SRT (step rate time); биты 4-7: head unload time
+1	1	бит 0: 1=исп.DMA; биты 2-7: head load time
+2	1	период отключения мотора (55-мс интервалов перед отключением мотора)
+3	1	Размер сектора (0=128, 1=256, 2=512, 3=1024)
44	1	номер последнего сектора на дорожке
+5	1	длина промежутка для операций чтения/записи
+6	1	максимальная длина данных
+7	1	длина промежутка для операции форматирования
+8	1	символ-заполнитель для форматирования (обычно 0f6H)
49	1	основной временной интервал в миллисекундах
+0aH	1	время запуска мотора (в 1/8-секундных интервалах)

Рис. 14.12. Параметры дискеты, расположенные по адресу, определяемому вектором 1ЕН.

Смещение	Длина	Содержание
+0	2	максимальное число цилиндров
+2	1	максимальное число головок
+3	2	starting reduced-write current cylinder
+5	2	starting write precompensation cylinder
+7	1	maximum ECC data burst length +- bit 7: disable disk-access retri
+8	1	drive step options _-¦ bit 6: disable ECC retries +- bits 2-0: drive option
+9	1	стандартное значение таймаута
+0aH	1	значение таймаута для форматирования
+0bH	1	значениетаймаута для контроля устройства
+0cH	4	(резерв)

Рис. 14.13. Параметры жесткого диска, расположенные по адресу, определяемому вектором 41Н. Если есть второй жесткий диск, то параметры определяет вектор 46Н.

V. Увеличение размера раздела и 32-битная FAT.

При увеличении раздела жесткого диска возникают проблемы с FAT. Можно увеличить ее размер. Но нельзя это делать безгранично. Во-первых, для большой FAT-таблицы требуется много места в памяти. Во-вторых, увеличивая таблицу, мы тем самым уменьшаем пространство для данных. Второй путь - увеличить размер кластера. Но здесь возникает проблема, связанная с тем, что в последних незаполненных кластерах теряется часть свободного пространства. Чем больше небольших файлов, тем больше эти потери. Такие потери могут достигать 40% всего свободного пространства.

В операционной системе Windows 95 появилась возможность создавать разделы с 32-битной FAT. Элемент в такой таблице имеет длину 4 байта. Это позволяет адресовать больше пространства и, следовательно, создавать большие разделы. Вместе с тем следует отметить, что 32-битная FAT - это полумера, т.к. опять закладывается некий предел (хотя и большой) на объем раздела. Но все это является следствием необходимости поддерживать совместимость с предыдущими версиями системы.

VI. Содержание списка списков.

О списке списков, получаемом при помощи функции 52Н, уже упоминалось. Сейчас приводится описание некоторых наиболее важных полей этой структуры.

Смещение	Длина	Описание
-8	4	указатель на текущий дисковый буфер
-2	2	сегментный адрес первого блока MCB
0	4	указатель на первый блок параметров диска
4	4	указатель на первую системную таблицу файлов
8	4	указатель на заголовок активного драйвера CLOCK$
12	4	указатель на заголовок активного драйвера CON
16	2	максимальный размер сектора в байтах
18	4	указатель на информационную запись дискового буфера
22	4	указатель на массив структур текущего каталога
32	1	число установленных блочных устройств
33	1	число элементов в массиве структур тек. каталога
34	18	заголовок драйвера устройства NUL
63	2	значение X в директиве BUFFERS=X,Y
65	2	значение Y в директиве BUFFERS=X,Y
67	1	дисковод загрузки

Поле со смещением 2 мы уже использовали для определения начала цепочки МСВ. В дальнейшем нам понадобится также начало системной таблицы файлов - смещение 4. Структура этой таблицы такова: вначале идут 6 байт заголовка. Первые 4 байта содержат ссылку на следующую таблицу или FFFFH, если таблица последняя. Далее идут блоки, структура и размер которых может, вообще говоря, меняться для различных версий MS DOS. Ниже представлены некоторые важные поля такого блока.

Смещение	Длина	Описание
0	2	количество дескрипторов, закрепленных за данным файлом, или 0, если блок свободен
2	2	режим доступа к файлу
4	1	атрибут файла
5	2	информационное слово устройства
7	4	указатель на заголовок драйвера символьного устройства или указатель на блок параметров дисковода
И	2	номер первого кластера файла
13	2	время последней модификации файла
15	2	дата последней модификации файла
17	4	размер файла в байтах
21	4	текущее положение указателя файла
25	2	относительный номер последнего прочитанного или записанного кластера
27	4	номер сектора с записью каталога о данном файле
31	1	номер записи каталога внутри сектора
32	11	имя и расширение файла
46	2	сегментный адрес PSP программы, открывшей файл
53	2	абсолютный номер последнего записанного или прочитанного кластера файла

В главе 19 будет показано, как последняя информация может быть использована для защиты программ от несанкционированного копирования.

VII. Соображения по поводу файловой структуры MS DOS.

Файловая система MS DOS, на мой взгляд, обладает рядом серьезных изъянов, которые я излагаю ниже, как обычно, пронумеровав их.

1. Ограничение на длину файловых имен. Причем не предусмотрено какого-либо резерва. Отсюда хитрости Windows 95 (см. ниже).

2. Ограничения, накладываемые на знаки, используемые для имен файлов. Сбой, в результате которого на диске появится файл с "экзотическим" именем, приводит к невозможности справиться с таким файлом средствами DOS.

3. Нет автоматической защиты от сбоев. Для устранения всяких неполадок приходится обращаться к утилитам типа Disk Doctor.

4. Нет защиты от прямого доступа к диску. Используя утилиту типа DISKEDIT.EXE, неискушенный пользователь легко может испортить содержимое диска. Я уже не говорю о неискушенном программисте, использующем прерывания 25H, 13H.

VIII. Длинные имена.

В Windows 95 реализована возможность использования для файлов длинных имен. Сейчас я намерен разобрать этот вопрос.

Естественно фирма Microsoft должна была подумать о совместимости с каталогами старого формата. Нужно было в каталогах старого формата (состоящих из 32-байтных записей) поместить информацию о длинном имени файла. При этом здесь же должны храниться записи старого формата.

Решение специалистов основывается на следующем факте: запись с атрибутом, равным OFH, не видна ни для каких функций DOS. Итак, что же нового привнесла Windows 95 в структуру каталогов?

1. Все файлы можно разделить на две группы: с обычными именами и с длинными именами.

2. Для файлов с обычными именами запись в каталоге осталась той же самой.

3. Файл с длинным именем имеет в каталоге одну запись, идентичную записи для файлов с обычным именем (первая запись), и несколько записей с атрибутом OFH, которые находятся перед обычной записью и в которых хранится длинное имя, а также другая информация (см. ниже).

4. Имя файла в первой записи - шесть первых символов его полного имени. Седьмым символом является ~, восьмым - цифра от 1 и выше. Цифра нужна, чтобы различать файлы, у которых шесть первых символов в имени совпадают.

Обратимся теперь к структуре записи для длинного имени. Отметим здесь две особенности:

1. Длинное имя хранится в UNICODE (на каждый символ 2 байта).

2. Длинное имя хранится в обратном порядке - начинается в записи ближней к первой записи, и заканчивается в самой дальней.

Структура:

байт 0 - байт порядка следования,

биты 0 - 4 содержат номер 1-31,

бит 5 = 0,

бит 6 - 1 - запись последняя (точнее, первая),

бит 7 = 0,

байты 1-10 - пять символов имени,

байт 11 - равен OFH (атрибут),
байт 12 - равен O,
байт 13 - контрольная сумма,
байты 14-25 - шесть символов имени,
байты 26-27 равны O,
байты 28-31 два символа имени.

Длинные имена в корневом каталоге могут создать определенные проблемы для пользователя. Дело в том, что размер корневого каталога в MS DOS фиксирован (см. выше). Windows 95 унаследовала этот недостаток. Ясно, что количество файлов с длинными именами в корневом каталоге ограничено этим размером.

Глава 15. Использование ассемблера с языками высокого уровня.

Какой он нации, сказать не знаю смело: на всех языках говорит...

М.Ю. Лермонтов
Маскарад.

Одним из удобств современных языков высокого уровня является встроенный ассемблер, который предоставляет возможность использовать язык ассемблера непосредственно в тексте программ на языке Си и Паскале. Это делается путем введения специальных ассемблерных блоков, с помощью ключевого слова ASM. Такой встроенный ассемблер довольно полно описывается в соответствующих руководствах по программированию, и мы не будем этим заниматься. Кстати сказать, встроенный ассемблер, как правило, отражает не все команды микропроцессора. Предмет же нашего разговора лежит в области классического интерфейса между различными языками, который основывается на существовании промежуточных файлов трансляции, которые называются объектными модулями (см. главу 13 и конец главы 1).

Вопрос стыковки различных языков друг с другом довольно интересен и сложен. Заинтересованного читателя отошлю к источнику [16]. Мы сужаем проблему и рассматриваем стыковку только ассемблера с языками высокого уровня. В качестве таковых берутся три языка (точнее, три компилятора): TURBO C++ (BORLAND), TURBO PASCAL (BORLAND) и QUICK BASIC[42] (MICROSOFT). Сказанное здесь, однако, в значительной степени будет относиться и к другим компиляторам, предполагающим интерфейс с языком ассемблера. Получив некоторые навыки, в дальнейшем Вы самостоятельно сможете использовать язык ассемблера с другими языками, предварительно просмотрев документацию по этим языкам.

I.

На Рис. 15.1 представлен модуль, процедура которого _CLRSCR может быть вызвана из программ на языках высокого уровня. В свою очередь, из этой процедуры вызывается процедура (или функция), находящаяся в основной программе. Эти программы представлены на Рис. 15.2, 15.3, 15.4. Приведу краткую инструкцию по объединению модулей.

1. Для того чтобы объединить указанный модуль с программой на языке BASIC (Рис. 15.2), удалите предварительно из него все подчеркивания. Далее оба модуля транслируются независимо и объединяются при помощи стандартного компоновщика LINK.EXE.
2. В случае программы на языке Паскаль следует вначале оттранслировать модуль. Транслятор Турбо Паскаля сам производит компоновку, поэтому далее достаточно выполнить команду: TPC MOD2/B.

---

[42] Речь, конечно, идет о компиляторе языка BASIC.

3. Транслятор с языка Си++ автоматически вызывает компоновщик TLINK.EXE. При трансляции используйте модель LARGE. Если же Вы хотите проводить компоновку отдельно, то следует правильно указать все объектные модули и библиотеки (см. конец главы Рис. 15.13).

```
        EXTRN _PR:FAR
CODE SEGMENT
        ASSUME CS:CODE
        PUBLIC _CLRSCR
_CLRSCR PROC FAR   ;процедура очистки экрана
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH ES
        MOV  AX,0B800H   ;полагаем этот адрес видеобуфера
        MOV  ES,AX
        MOV  CX,2000
        XOR  BX,BX
LOO:
        MOV  ES:[BX],0700H
        ADD  BX,2
        LOOP LOO
        POP  ES
        POP  CX
        POP  BX
        POP  AX
        CALL FAR PTR _PR
        RET
_CLRSCR  ENDP
CODE   ENDS
        END
```

Рис. 15.1. Модуль на языке ассемблера, который может быть подключен к модулям на Рис. 15.2, 15.3, 15.4. Для подсоединения к модулю на языке ***B****ASIC* ***следует*** *убрать все подчеркивания.*

```
DECLARE  SUB  CLRSCR
        CALL  CLRSCR
END
SUB  PR  STATIC
        PRINT"ПРОЦЕДУРА В ПРОГРАММЕ НА QUICK BASIC"
ENDSUB
```

Рис. 15.2. Программа ***на языке*** *QUICKBASIC.*

```
{интерфейсный модуль MOD1.PAS на языке Турбо Паскаль}
UNIT
MOD1;
INTERFACE
PROCEDURE _PR;
PROCEDURE _CLRSCR;
IMPLEMENTATION
{$L BLOK}
PROCEDURE _CLRSCR; EXTERNAL;
PROCEDURE _PR;
BEGIN
      WRITELN('ПРОЦЕДУРА В ПРОГРАММЕ НА TURBO PASCAL');
END;
END.

{основной модуль MOD2.PAS на языке TURBO PASCAL}
USES MOD1;
BEGIN
      _CLRSCR;
END.
```

Рис. 15.3. Модули на языке Турбо Паскаль.

```
#INCLUDE <STDIO.H>EXTERN VOID CLRSCR(VOID);VOID  PR(VOID);

VOID PR(VOID)
{
      CHAR S[36]="ПРОЦЕДУРА В ПРОГРАММЕ НА TURBO CI++\N";
      PUTS(S);
}

VOID MAIN(VOID)
{
       CLRSCR();
}
```

Рис. 15.4. Программа на языке Си.

Я думаю, что читатель без особого труда разберется в программах на Рис. 15.1 - 15.2. Однако есть некоторые нюансы, о которых стоит поговорить. Мне не хотелось бы давать некоторые исчерпывающие правила. Попробуем порассуждать о том, почему может не получиться компоновка таких программ. В данном разделе мы не касаемся вопроса передачи параметров и использования общих данных.

Итак, Вы написали два модуля (конечно, их может быть больше) - один на языке ассемблера, а другой на языке высокого уровня. Как положено, все общие процедуры

Вы отметили как PUBLIC и EXTRN[43]. Однако программа не идет. Либо появляется ошибка при компоновке, либо ошибка проявляется во время работы программы. Проблема может быть в следующем:

1. Неправильно осуществляется вызов процедуры, находящейся в другом модуле.
2. Неправильно осуществляется возврат из такой процедуры.

Это ключевые моменты - третьего не дано. Рассмотрим вначале пример на Паскале. Замечу, что согласно описанию Турбо Паскаля на ассемблерный код накладываются жесткие требования по поводу имен кодовых сегментов. Кодовый сегмент может иметь имена CODE, CSEG или оканчиваться на _TEXT. Это условие было, естественно, выполнено. Причем компилятор не требует указания ни типов, ни классов этих сегментов. Компилятор не формирует самостоятельных сегментов (в отличие от Си и Бейсика), а объединяет их с модулями, где данные процедуры были объявлены. Согласование вызовов и возвратов осуществлено тем, что, во-первых, в ассемблерном модуле и процедура _PR, и процедура _CLRSCR имеют атрибут FAR, во-вторых, в модуле MOD1.PAS обе эти процедуры указаны в разделе INTERFACE. Проделаем теперь такой эксперимент: уберем объявление процедуры _PR из интерфейсного раздела. Трансляция программы завершается успешно, однако возврата из процедуры _PR не происходит. Выход из этой процедуры происходит по RETN, а не по RETF, как раньше. Вызов же производится длинным CALL. Вы можете исправить ассемблерный модуль, заменив атрибуты FAR у _PR на NEAR. После этого все пойдет нормально. Похожая ситуация возникнет в том случае, если мы объявим _CLRSCR в главном модуле. Теперь вызов к ней будет близким, но возврат дальним. Здесь аналогично проблема разрешается изменением в ассемблерном коде атрибута у _CLRSCR (с FAR на NEAR).

В случае с Си ситуация сложнее, но и гибче одновременно. Работая с моделями памяти MEDIUM и LARGE, Вы можете брать любые имена для кодовых сегментов в ассемблерных модулях. При этом, естественно, все вызовы должны быть длинными. В программе на Си при этом не обязательно указывать атрибуты (FAR или NEAR), т.к. они берутся по умолчанию. Измените модель на SMALL, и Вы не сможете скомпоновать программу. Для того чтобы решить проблему, вспомним, что модель SMALL имеет всего один кодовый сегмент. Кроме того, в этой модели используются близкие вызовы. С учетом этого измените в ассемблерном коде все атрибуты с FAR на NEAR, а заголовок сегмента измените на _TEXT SEGMENT BYTE PUBLIC 'CODE'. Дело в том, что именно такой сегмент (такого имени, типа и класса) формируется транслятором Си. При компоновке оба сегмента, естественно, объединятся. Вы можете объединить сегменты и в модели LARGE. Если Вы объедините два сегмента, коды которых работают только друг с другом, то сможете смело изменить все атрибуты в ассемблерном коде с FAR на NEAR. Не забудьте при этом в программе на языке Си в соответствующих объявлениях поставить атрибут NEAR. Для того чтобы выяснить, какие сегменты и как следует объединять, используйте ключ -S при трансляции. Вы получи-

[43] В языках высокого уровня вместо PUBLIC и EXTERN используются другие средства.

те ассемблерный код из программы на Си, просматривая который, Вы узнаете все, что необходимо, из первых рук.

Ситуация с Бейсиком похожа на предыдущую ситуацию, но проще. Используйте любое имя для ассемблерного сегмента и дальние переходы[44]. Успех будет сопутствовать Вам.

П.

В модуле на языке ассемблера Вы можете определять и хранить данные. Разумеется, данные можно поместить как непосредственно в кодовый сегмент, так и создать для этого отдельный сегмент данных. Однако из языка высокого уровня доступ к этим данным будет закрыт.

Проблему легко разрешить, если Вы знаете, в каком сегменте помещает данные компилятор языка высокого уровня. Если это известно, то в ассемблерном модуле можно определить нужный сегмент и получить доступ к данным в этом сегменте. Конечно, кроме этого, Вы должны знать форматы данных для данного языка. Узнать все форматы можно из соответствующей документации.

Доступ к данным в программе на языке Турбо Паскаль можно получить через сегменты CONST (константы), или сегмент, оканчивающийся на _DATA (инициализированные переменные), или через сегменты с именами DATA, DSEG и сегменты с именами, оканчивающимися на _BSS. На Рис. 15.5 показан пример, демонстрирующий сказанное. В основном модуле на Паскале задается строковая переменная, а печатается в модуле на ассемблере. Обратите внимание, как задан сегмент данных и внешняя переменная STR. Заметим, что здесь понадобилась информация о том, в каком формате Турбо Паскаль хранит строковые переменные (последовательность кодов ASCII, перед которой стоит длина строки).

На Рис. 15.6 представлен аналогичный пример для языка Си++. Различие заключается лишь в формате строковых переменных. Для языка Си признаком конца строки служит символ с кодом 0. При компоновке используйте модель LARGE или MEDIUM.

С Бейсиком дела обстоят несколько сложнее. Оказывается, компилятор языка Бейсик фирмы Микрософт не генерирует внешние имена для переменных. Поэтому бессмысленно пытаться определить внешние имена в ассемблерном модуле, соответствующие переменным в модуле на Бейсике. Однако можно обойтись и без этого, если знать, в каком сегменте располагаются переменные. Для Бейсика это сегмент с именем BC_DATA и классом BC_VARS (тип выравнивания у сегмента данных обычно WORD). Конечно, трудно искать вслепую, но если знать порядок инициализации переменных и их формат, то это вполне допустимый подход. Можно упростить ситуацию, если использовать ключевое слово COMMON. Переменные, определенные с помощью этого слова, помещаются в сегмент с этим именем и классом BLANK. Теперь достаточно определить с помощью COMMON те переменные, которые мы хотим менять в ассемблерном модуле, а дальше уже дело техники. Рис. 15.7 иллюстрирует сказанное выше. В ассемблерном модуле происходит изменение значений двух целочисленных переменных AA и BB.

[44] В языке BASIC MICROSOFT для вызовов используются только дальние адреса.

```
;модуль на языке ассемблера
DATA SEGMENT WORD 'DATA'
EXTRN STR:NEAR
DATA ENDS
PUBLIC PRINT;
CODE SEGMENT
      ASSUME CS:CODE, DS:DATA
PRINT PROC FAR ;процедура вывода строки на экран
      PUSH DS
      PUSH AX
      PUSH CX
      PUSH BX
      MOV  AX,SEG STR
      MOV  DS,AX
      XOR  CH,CH
      MOV  CL,BYTE PTR DS:STR
      MOV  BX,OFFSET DS:STR+1
      MOV  AH,2
LOO:
      MOV  DL,DS:[BX]
      INT  21H
      INC  BX
      LOOP LOO
      POP  BX
      POP  CX
      POP  AX
      POP  DS
      RET
PRINT ENDP
CODE ENDS
      END

{модуль 1 на языке Паскаль}
UNIT MOD1;
INTERFACE
VAR
 STR:STRING;
PROCEDURE PRINT;
IMPLEMENTATION
{$L BLOK}
PROCEDURE PRINT; EXTERNAL;
END.
```

```
{модуль 2 на языке Паскаль}
USES MOD1;
BEGIN
      STR:='ПЕЧАТАЕМ ИЗ МОДУЛЯ НА ЯЗЫКЕ АССЕМБЛЕРА';
      PRINT;
END.
```

Рис. 15.5. Доступ к общим данным при компоновке модулей на языке ассемблера и на языке Turbo Pascal.

```
;модуль на языке ассемблера
_DATA SEGMENT BYTE PUBLIC 'DATA'
EXTRN _STR:BYTE
_DATA ENDS
PUBLIC _PRINT
CODE SEGMENT
      ASSUME CS:CODE, DS:_DATA
_PRINT PROC FAR  ; процедура вывода строки на экран
      PUSH DS
      PUSH AX
      PUSH BX
      MOV  AX,SEG _STR
      MOV  DS,AX
      MOV  BX,OFFSET DS:_STR
      MOV  AH,2
LOO:
      CMP  BYTE PTR DS:[BX],0
      JZ   _END
      MOV  DL,DS: [BX]
      INT  21H
      INC  BX
      JMP  SHORT LOO
_END:
      POP  BX
      POP  AX
      POP  DS
      RET
_PRINT ENDP
CODE ENDS
      END
```

```
/*модуль на языке Си*/
EXTERN VOID  PRINT(VOID);
CHAR STR[]="ПЕЧАТАЮ ИЗ МОДУЛЯ НА ЯЗЫКЕ АССЕМБЛЕРА.";

VOID MAIN(VOID)
{
      PRINT ();
}
```

Рис. 15.6. Доступ к общим данным при компоновке модулей на языке ассемблера и на языке Турбо Си++.

```
;модуль на языке ассемблера
COMMON SEGMENT BYTE PUBLIC 'BLANK'
COMMON ENDS
PUBLIC INASM
CODE SEGMENT
      ASSUME CS:CODE
INASM PROC FAR
      PUSH DS
      PUSH AX
      MOV AX,COMMON
      MOV DS,AX
      MOV WORD PTR DS:[0],1234    ;переменная AA
      MOV WORD PTR DS:[2],456     ;переменная BB
      POP AX
      POP DS
      RET
INASM ENDP
CODE  ENDS
      END
```

```
'модуль на языке Бейсик
DECLARE SUB INASM
COMMON SHARED AA AS INTEGER
COMMON SHARED BB AS INTEGER
AA=23
BB=990
PRINT AA,BB
CALL INASM
PRINT AA,BB
END
```

Рис. 15.7. Доступ к общим данным при компоновке модулей на языке ассемблера и на языке Бейсик Микрософт.

III.

Перейдем к вопросу о передаче параметров в процедуру и из нее. Этот вопрос неплохо был разобран в главе 13, теперь закрепим пройденное. Если мы обратимся к ассемблерному коду программы, написанной скажем на Си (это несложно сделать, используя параметр S), то легко выявить общность в структуре процедур. Эта общая структура изображена на Рис. 15.8. Структура стека во время выполнения данной процедуры изображена на Рис. 15.9. Обращаю Ваше внимание на то, что при выходе из процедуры в стеке еще остаются параметры (если они были). В случае языка Паскаль или Бейсик возврат нужно осуществлять по RET K, где K - размер области параметров. В Си об этом заботиться не надо, т.к. по выходу из функции транслятор автоматически освобождает

стек (SP-K). Когда Вы пишите процедуру для компоновки с программой на языке высокого уровня, Вам следует придерживаться изложенного подхода.

Доступ к параметрам и локальным переменным осуществляется через регистр BP. Легко видеть, что в случае дальней процедуры на область параметров указывает [BP+6], а на область локальных переменных (в Ваших процедурах на языке ассемблера их скорее всего не будет) [BP-2] (на первое слово). В случае близкой процедуры на параметры будет указывать [BP+4]. Запомните, что в любом случае перед выходом из процедуры должны быть восстановлены 4 регистра - DS, SS, SP, BP.

Параметры могут передаваться по значению и по указателю. Указатель при этом может быть как длинным, так и коротким. Длинный указатель соответствует паре **сегмент:смещение,** а короткий - равен смещению (предполагается, что берется сегмент, на который в данный момент показывает DS).

```
ИМЯ_PROC  PROC  FAR
        PUSH  BP
        MOV   BP,SP
        SUB   SP, РАЗМ._ЛОК. ;выделяем место для локальных
                             ;переменных
        {сохраняем нужные регистры}
        {выполняем необходимые действия}
        {восстанавливаем регистры}
        ADD SP,РАЗМ._ЛОК    ;удаляем локальные переменные
        POP  BP
        RET
ИМЯ_PROC  ENDP
```

Рис. 15.8. Общая структура процедуры в языке высокого уровня.

Младшие адреса

Сохраненные значения регистров	← SP
Область локальных переменных	
Сохраненное значение BP	← BP
Адрес возврата	
Область параметров	

Старшие адреса

Рис. 15.9. Структура стека во время работы процедуры.

Хочу заметить, что команда SUB SP,РАЗМ._ЛОК. перемещает указатель стека так, что рост стека не может испортить локальные переменные или параметры. Возможна, однако, и другая альтернатива: указатель стека поместить не в меньшие, а в большие адреса. Здесь, однако, есть опасность, что указатель стека будет указывать на какие-либо данные, которые оказались в стеке при вызове процедуры, из которой была вызвана данная. В этом случае компилятор должен иметь информацию об объеме использованного уже стека.

Теперь, получив некоторый теоретический заряд, мы займемся примерами. Первый пример будет касаться языка Паскаль. В Паскале возможна передача как по ссылке, так и по значению. Параметры переменные всегда передаются по ссылке. Параметры значения передаются по значению, если их длина не превышает 4 байта. Параметры значения длиной свыше 4 байт передаются по ссылке на некоторую локальную область, куда этот параметр предварительно должен быть помещен. Порядок проталкивания в стек параметров процедуры - слева на право. Например, если вызывается процедура SUM(A,B:WORD), то вначале в стек будет помещен параметр А, а затем В. Что касается ссылок, то в Паскале передаваемые в процедуру ссылки всегда длинные.

```
;ассемблерный модуль
PUBLIC  PRO
_TEXT SEGMENT
        ASSUME CS:_TEXT
PRO  PROC
        PUSH  BP
        MOV   BP,SP
        PUSH  AX
        PUSH  DX
        PUSH  DS
        PUSH  BX
        LDS   BX,[BP+4]                 ;*
        MOV   DL,[BX]                   ;*
        MOV   AH,2                      ;*
        INT   21H                       ;*
        MOV   BYTE PTR  [BX],65         ;*
        LDS   BX,[BP+8]                 ;*
        MOV   DL,[BX]                   ;*
        INT   21H                       ;*
        POP   BX
        POP   DS
        POP   DX
        POP   AX
        POP   BP
        RET,  8
PRO ENDP
_TEXT  ENDS
        END
```

```
{модуль на языке Турбо Паскаль}
VAR
 A,B:BYTE;
 I:LONGINT;
{$L BLOK}
PROCEDURE PRO(VAR A,B:BYTE); EXTERNAL;
BEGIN
      A:=65; B:=66;
      PRO(A,B);
      WRITELN;
      WRITELN(CHR(A),CHR(B));
END.
```

Рис. 15.10. Пример передачи параметров в процедуру на ассемблере из программы на языке Паскаль.

Рассмотрим пример на Рис. 15.10. Здесь два параметра из программы на Паскале передаются в процедуру PRO, расположенную в ассемблерном модуле. Вызов процедуры близкий, однако, параметры передаются по длинной ссылке. Параметр B меняется в процедуре PRO, что "чувствует" основная программа: в первой строке будет напечатано BA, тогда как во второй строке - AA. Если теперь параметры A и B будут параметрами значениями, то это потребует некоторого изменения процедуры PRO. Строки, отмеченные звездочкой, следует заменить на

```
MOV DL,[BP+4]
MOV AH,2
INT 21H
MOV BYTE PTR [BP+4],65
MOV DL,[BP+6]
INT 21H
```

В конце программы следует поставить RET 4, т.к. отсутствует команда передачи в стек байта - вместо этого передается целое слово. В этом случае программа не "почувствует" изменения параметра B.

Работа с функциями на Паскале аналогична работе с процедурами. Здесь следует только знать, как и куда должно возвращаться значение функции. В документации по программированию на Паскале, Вы без труда найдете эти сведения.

Обратимся к Бейсику. Здесь необходимо запомнить следующие правила:

1. В языке Бейсик Микрософт по умолчанию все параметры передаются по близкой ссылке.
2. Параметр передается по значению, если вызываемая подпрограмма объявлена при помощи оператора DECLARE и к параметру применено ключевое слово BYVAL. Использование ключевого слова CALLS заменяет эту принимаемую по умолчанию установку, и параметры передаются по удаленной ссылке. Применение к параметру ключевого слова SEG также приводит к тому, что он будет передаваться по удаленной ссылке.

```
;ассемблерный модуль
PUBLIC  INASM
EXTRN  INBAS:FAR
CODE  SEGMENT
       ASSUME  CS:CODE
INASM  PROC  FAR
       PUSH  BP
       MOV   BP,SP
       PUSH  AX
       PUSH  BX
       PUSH  DS
       LDS   BX,[BP+6]
       ADD   WORD  PTR  [BX],10
       LDS   BX,[BP+10]
       ADD  WORD  PTR  [BX] , 10
       PUSH  WORD  PTR  [BP+10]
       PUSH  WORD  PTR  [BP+6]
       CALL  INBAS
       POP   DS
       POP   BX
       POP   AX
       POP   BP
       RET  8
INASM  ENDP
CODE   ENDS
       END

'модуль на языке Бейсик
DECLARE  SUB  INASM(SEG  A1%,SEG  A2%)
A%=13:B%=2623
PRINT  A%,B%
CALL  INASM(A%,B%)
END
SUB  INBAS(X1%,X2%)
  PRINT  X1%,X2%
END  SUB
```

Рис. 15.11. Пример обмена параметрами, между процедурой па ассемблере и программой на Бейсике.

Рассмотрим пример на Рис. 15.11. *В* этом примере производится обмен параметрами между основной программой, написанной на Бейсике, и процедурой, написанной на ассемблере. Если в процедуру на ассемблере параметры пересылаются по длинной ссылке, то в процедуру INBAS они приходят (предварительно увеличенные на 10) по короткой ссылке. Вызов процедур INASM и INBAS также длинный.

В Си ситуация в целом похожая. Надо только помнить, что последовательность передачи параметров здесь обратная - параметры вталкиваются в стек справа налево. Параметр может быть как значением, так и ссылкой - в зависимости от его типа. Ссылка может быть как далекой, так близкой. По умолчанию все зависит от выбранной модели памяти. Например, для модели SMALL ссылки будут близкими, а для модели LARGE - дальними. Изменить тип ссылки можно при помощи ключевых слов NEAR и FAR.

```
;ассемблерный модуль
PUBLIC  _INASM
EXTRN   _INCI:FAR
CODE  SEGMENT
        ASSUME  CS:CODE
I       DW  0
J       DW  0
_INASM  PROC  FAR
        PUSH  BP
        MOV   BP,SP
        PUSH  DS
        PUSH  BX
        PUSH  CS
        POP   DS
        MOV   AX, [BP+6]
        SHR   AX,1
        MOV   I,AX
        MOV   J,10000
        PUSH  DS
        MOV   AX,OFFSET I
        PUSH  AX
        PUSH  DS
        MOV   AX,OFFSET J
        PUSH  AX
        CALL  _INCI             ;в AX значение функции
        ADD   SP,8              ;восстановили стек
        POP   BX
        POP   DS
        POP   BP
        RET
_INASM  ENDP
CODE  ENDS
        END

/*модуль на языке Си*/
EXTERN  INT  INASM(INT  I);
INT  INCI(INT * I,INT * J);
```

```
VOID MAIN(VOID)
{
      PRINTF("%D",INASM(30000));
}
INT INCI(INT *I,INT *J)
{
      RETURN((*I)+(*J));
}
```

Рис. 15.12. Пример обмена параметрами между процедурой на ассемблере и программой на Бейсике.

Параметр в программе на Рис. 15.12 проходит ряд превращений. Вначале он посылается как значение в процедуру на ассемблере. Там это значение делится на 2 и пересылается по далекой ссылке вместе с еще одним параметром в функцию, находящуюся в модуле на Си. В этой функции значения складываются, и результат отсылается обратно. Наконец, оттуда это значение опять переходит в основной модуль и там печатается. Напомню, что значение функции размером, не превышающим слово, возвращается через регистр AX (это справедливо и для других языков). Напомню также, что восстановление стека в Си происходит после возвращения из процедуры. Различие в порядке передачи параметров в языках Си и, скажем, Паскаль является достаточно существенным. Вдумайтесь: при передаче параметров в процедуру в языке Паскаль процедура получает доступ вначале к последнему параметру. Другими словами, она должна "знать", сколько параметров ей передается. В языке Си все наоборот: Вы не заботитесь заранее, сколько параметров предполагается передать. Этому же способствует тот факт, что очистка стека от параметров происходит не в процедуре (она же не "знает" реальное состояние стека), а в основной программе.

В Паскале существует такое понятие, как вложенные процедуры. Отличие вызова вложенной процедуры от обычной процедуры заключается лишь в том, что в стек вместе с параметрами проталкивается значение регистра BP. Таким образом вложенная процедура получает доступ к локальным переменным вызывающей процедуры. С точки зрения программирования вложенные процедуры, на мой взгляд, не представляют никакого удобства.

IV.

После прочтения данной главы Ваши возможности как программиста, несомненно, расширятся. Причем как за счет умения использовать ассемблер в программах на языке высокого уровня, так и наоборот, что иногда эффективнее и важнее. Конечно, можно писать основной модуль на ассемблере и обращаться время от времени к процедурам языка высокого уровня. Однако такой подход несколько утомителен, так как необходимо разбираться в структуре библиотечных процедур. Проще поступать так: основным модулем считать модуль, скажем, на языке Си. Однако этот модуль содержит только вызов некоторой ассемблерной процедуры, а также функции, которые вызываются из этой процедуры. Вся программа будет написана на ас-

семблере, а наиболее сложные операции будут выполняться мощными и эффективными функциями языка Си.

Несколько слов следует сказать о том, какие регистры следует сохранять в ассемблерных процедурах, написанных для вызова их из языков высокого уровня. Как Вы уже поняли, сохранять необходимо BP в силу той роли, которую он играет. Также очевидно, что нельзя портить регистры DS и SS. Для языка Паскаль и Бейсик список сохраняемых регистров этим и ограничивается. Для языка Си всегда существовал режим компиляции, когда для хранения переменных могут использоваться и регистры. Для этих целей используются два регистра SI и DI. Вот о них и следует также позаботиться, если Вы разрешаете компилятору хранить переменные в регистрах.

V.

В этом разделе несколько противоречивым выглядит то, что использование библиотечных процедур языков высокого уровня в языке ассемблера вполне возможно, а иногда и удобно. Ниже приводится пример вызова стандартной библиотечной функции языка Си: _CLRSCR() - очистки экрана из программы на языке ассемблера. Заметим, что ничего от фирмы BORLAND, кроме, разумеется, библиотеки, использовать необязательно.

```
;программа CLR.ASM
EXTRN _CLRSCR:FAR
PUBLIC _MAIN                    ;для связи с модулем COL.OBJ
_DATA SEGMENT
_DATA ENDS
STK SEGMENT STACK
      DB 100 DUP(?)
STK ENDS
_TEXT SEGMENT BYTE PUBLIC 'CODE'
      ASSUME CS:_TEXT,DS:_DATA,SS:STK
_MAIN PROC FAR
      CALL FAR PTR _CLRSCR ;вызов функции очистки экрана
;выход из программы, можно просто RET
      MOV AH,4CH
      INT 21H
_MAIN ENDP
_TEXT  ENDS
      END
```

Рис. 15.13. Программа на ассемблере, использующая стандартную функцию языка Си.

После трансляции данной программы мы получим объектный модуль CLR.OBJ. Следующая задача — это правильно скомпоновать его. Для компоновки используем стандартную библиотеку Си CL.LIB (дальняя модель) и объектный модуль COL.OBJ, необходимый для создания загрузочного модуля (вначале работают процедуры из

COL.OBJ, а затем передается управление на _MAIN). В командной строке пишем: LINK COL+CLR. На вопрос о библиотеке указываем CL. После этого программа очистки экрана будет получена.

```
;программа PUTS.ASM
EXTRN _PUTS:FAR
PUBLIC _MAIN          ;для связи с модулем COL.OBJ
_DATA SEGMENT BYTE PUBLIC 'DATA'
;данная строка будет напечатана
S DB 'Проверка.',0
_DATA ENDS
STK SEGMENT STACK
      DB 100 DUP(?)
STK ENDS
_TEXT SEGMENT BYTE PUBLIC 'CODE'
      ASSUME CS:_TEXT,DS:_DATA,SS:STK
_MAIN PROC FAR
;адрес выводимой строки
      PUSH DS
      LEA AX, S
      PUSH AX
      CALL FAR PTR _PUTS ;вызов функции очистки экрана
;освобождаем стек
      POP DX
      POP DX
;выход из программы, можно просто RET
      MOV AH,4CH
      INT 21H
_MAIN ENDP
_TEXT  ENDS
      END
```

Рис. 15.14. Пример с передачей параметров.

Пример на Рис. 15.14характерен тем, что вызывается функция с передачей параметров. В функцию передается адрес строки. Согласно принятому в Си соглашению освобождение стека происходит не в вызванной функции, а после возврата из нее. Последовательность двух команд POP DX и предназначена для этой цели.

VI.

Повсеместная популярность языков баз данных побуждает меня дать хотя бы краткое описание использования ассемблера в этой области. Для определенности буду рассматривать семейство DBASE, причем в его командном, а не компилируемом варианте.

Основой использования ассемблера в базах данных служат две команды:

```
LOAD имя  *указывается имя двоичного файла,  по умолчанию
          *расширение BIN
CALL имя  [WITH параметры]
```

Имя в команде CALL должно совпадать с именем загружаемого файла. Всего может быть загружено несколько файлов, и каждый получит свою область памяти. Освободить память можно с помощью команды RELEASE. Структура ассемблерного модуля совпадает со структурой COM-файла с той лишь разницей, что вместо ORG 100H следует поставить ORG 0. На передаваемые в модуль параметры указывает DS:[BX]. Если содержимое BX равно 0, то это означает, что параметров нет. Возврат из ассемблерного модуля должен быть длинным. Не пытайтесь в ассемблерном модуле менять структуру данных, т.к. можно испортить значение некоторых переменных.

Ниже на Рис. 15.15 представлен ассемблерный модуль, а также фрагмент программы, вызывающей этот модуль.

```
CODE  SEGMENT
      ORG 0
      ASSUME CS:CODE
      CMP BX,0
      JZ   EXIT    ;нет параметров
      MOV AH,2L1:
      MOV DL,DS:[BX]
      CMP DL,0
      JZ   EXIT    ;конец строки?
      INT 21H
      INC BX
      JMP SHORT L1
;--передать управление исполняющей системе--
EXIT:
      RETF
CODE  ENDS
      END
```

а) ассемблерный модуль

```
AA='Печать строки.'
LOAD WWW         *вызов модуля WWW.BIN
CALL WWW WITH AA
```

б) фрагмент программы вызова ассемблерного модуля

Рис. 15.15. Пример ассемблерногомодуля и вызова его на языке FOXBASE.

При разборе примера на Рис. 15.15 следует обратить внимание на формат строки в языке FOXBASE. Как видим, формат совпадает с форматом строк языка Си.

VII.

Одним из моих любимых занятий является исследование того, как отдельные фрагменты на языке высокого уровня преобразуются в ассемблерный код. Поверьте, это одно из самых поучительных и полезных занятий. Наиболее удобно работать в этом плане с Турбо Си. Здесь у компилятора имеется опция, которая заставляет его генерировать ассемблерный код. Впрочем, можно поступить и по-другому: откомпилировать программу с отладочной информацией, а потом использовать стандартный отладчик (Турбо DEBUGER для компиляторов фирмы BORLAND и CODEVIEW для Микросовтовских компиляторов). Рассмотрим несколько примеров.

Многие считают, что использование переменных типа BYTE (UNSIGNED CHAR) позволяет несколько оптимизировать работу программы (в пику использованию переменных типа INT). Рассмотрим, всегда ли так бывает. Ниже представлен фрагмент программы на Турбо Си.

```
INT I=1;
WHILE(I<=10) {
C[I]='H';
      I++;
}
```

Что же произойдет при преобразовании его в ассемблерный код? У Турбо Си имеется опция: использовать регистровые переменные тогда, когда это возможно. Это существенный момент. Переменная I будет помещена в регистр SI. Смотрим ассемблерный код.

```
      MOV    SI,1          ;INT I=1
      JMP    SHORT L1
L2:
      MOV    BYTE PTR [BP+SI-10],72 ;C[I]='H'
      INC    SI            ;I++
L1:
      CMP    SI,10
      JLE    SHORT L2
```

Ассемблерный фрагмент довольно прозрачен и в комментарии не нуждается. Что же произойдет, если мы I определим как UNSIGNED CHAR (т.е. BYTE). Смотрим фрагмент.

```
      MOV    CL,1          ;UNSIGNED CHAR I=1
      JMP    SHORT L1
L2:
      MOV    AL,CL
      MOV    AH,0
      LEA    DX,WORD PTR [BP-10]
      ADD    AX,DX
      MOV    BX,AX
```

```
        MOV     BYTE PTR SS:[BX],72     ;C[I]='H'
        INC     CL                      ;I++
L1:
        CMP     CL,10
        JBE     SHORT L2
```

Ну что, Вы удивлены? Вот именно! Использование байтовой переменной в данном случае увеличило код программы и, очевидно, увеличило время ее работы. Отменим теперь использование регистровых переменных и посмотрим ассемблерный фрагмент для случая INT I=1.

```
        MOV     WORD PTR [BP-12],1   ;INTI=1
        JMP     SHORT L1
L2:
        LEA     AX,WORD PTR [BP-10]
        MOV     BX,WORD PTR [BP-12]
        ADD     BX,AX
        MOV     BYTE PTR SS:[BX],72  ;C[I]='H'
        INC     WORD PTR [BP-12]     ;I++
L1:
        CMP     WORD PTR [BP-12],10
        JLE     SHORT L2
```

Ниже представлен фрагмент для UNSIGNED CHAR I=1.

```
        MOV     BYTE PTR [BP-11],1  ;UNSIGNED CHAR I=1
        JMP     SHORT L1
L2:
        MOV     AL,BYTE PTR [BP-11]
        MOV     AH, 0
        LEA     DX,WORD PTR [BP-10]
        ADD     AX,DX
        MOV     BX,AX
        MOV     BYTE PTR SS:[BX],72 ;C[I]='H'
        INC     BYTE PTR [BP-11]    ;I++
L1:
        CMP     BYTE PTR [BP-11],10
        JBE     SHORT L2
```

Как видите, в этом случае фрагменты с точки зрения оптимальности достаточно близки друг к другу. Все же я бы предпочел опять выбрать случай INT I=1. Примерно также обстоит дело в случае аналогичной программы на Паскале. Определение переменной I как байтовой величины приводит к некоторому усложнению результирующего кода.

Еще один интересный пример. В книжках по Си часто приходится читать, что использование указателей предпочтительнее, чем индексов. Забегая вперед, хочу согласиться с этим утверждением и представить доказательство этому положению. Рассмотрим два фрагмента на Си.

Фрагмент 1.

```
LONG C[10];
INT I;
FOR(I=0; I<10; I++) C[I]=1;
```

Фрагмент 2.

```
LONG C[10];
LONG * S;
INT I;
S=&C[0];
FOR(I=0; I<10; I++) *(S++)=1;
```

Ниже представлены ассемблерные коды, соответствующие циклам этих фрагментов.

Для фрагмента 1.

```
        XOR     DX,DX
        JMP     SHORT L1
L2:
        MOV     BX,DX           ;получить
        MOV     CL,2            ;значение
        SHL     BX,CL           ;индекса
        LEA     AX,WORD PTR [BP-40]
        ADD     BX,AX            ;и добавить к базовому адресу
        MOV     WORD PTR SS:[BX+2],0
        MOV     WORD PTR SS:[BX],1
        INC     DX
L1:
        CMP     DX,10
        JL      SHORT L2
```

Для фрагмента 2.

```
        XOR     AX,AX
        JMP     SHORT L1
L2:
        LES     BX,DWORD PTR [BP-4]
        MOV     WORD PTR ES:[BX+2],0
        MOV     WORD PTR ES:[BX],1
```

```
        ADD WORD  PTR [BP-4],4   ;увеличить значение указателя
                                 ;сразу на 4
        INC       AX
L1:
        CMP       AX,10
        JL        SHORT  L2
```

Как можно заметить, ассемблерный фрагмент во втором случае более эффективен. И здесь все ясно: указатель непосредственно определяет элемент, а индекс, прежде чем получить элемент, на который он указывает, должен быть подвергнут раду преобразований. В данном примере индекс вначале умножается на 4, а затем добавляется к базовому адресу. Я полагаю, что из уже приведенных фрагментов, читатель должен извлечь еще одну мораль: на ассемблере программу можно сделать гораздо короче.

Ассемблер вошел в мою кровь, когда мне приходилось программировать на маленьких компьютерах. Согласитесь, что если на компьютере всего 64 килобайт оперативной памяти, то о языках высокого уровня говорить не приходится. Впоследствии жизнь заставила меня программировать на разных языках. На мой взгляд, в языке программирования высокого уровня должен быть соблюден баланс между удобством программирования и пониманием того, как работает программа на низком уровне. Поясню свою мысль. Рассмотрим такое понятие, как "указатель". Оно является основополагающим для такого языка, как Си. Однако стоило ли отделять его от понятия адреса. Адрес можно записать в различных формах, но все равно это будет адрес. Но, введя понятия указателя, я думаю, авторы чисто психологически посчитали разумным различать указатели по тому, на какой объект они направлены (переменные различных типов, функции, структуры и т.д.). Конечно, они полагали удобным, когда компилятор сможет контролировать типы входных параметров. Писать программы для программиста средней руки, возможно, стало легче, но зато затруднило понимание механизма работы самой программы. А между тем нет никакой разницы между адресом строки байт и адресом функции. По сути, все, что мы имеем в памяти, это лишь цепочки байт.

В умных книжках по Си много говорится на тему, как эффективнее программировать. При этом часто смешиваются такие понятия, как эффективный код и эффективный текст. Мы же, как читатель уже догадывается, печемся в основном об эффективности кода. Рассмотрим несколько простых примеров.

1. Чем отличается условие if(!a) от условия if(a==0).Прочтя умные книжки, наивный читатель решит, что первое условие более эффективно. Ничуть не бывало - они совершенно одинаковы. Если переменная "а" будет помещена в регистр, ну, скажем, DI, то проверка условия сведется к командам:

```
or di,di
jne 11    и т.д.
```

   причем и в том, и в другом случае. Но второй случай явно предпочтительнее, т.к. улучшает читаемость программы. Вот так, "зри в корень", - говаривал Козьма Прутков.

2. Недавно прочел еще одну удивительную вещь. Автор предлагает, где это возможно, заменять оператор «if» оператором «?». Ну, скажем, в таком случае:

```
if(j==2) i=j; else i=k;    на    i=(j==2)?(j):(k);
```

Наивный автор не понимает, что во втором случае фактически записана функция, и компилятор сгенерирует код, в котором результат вначале будет помещен в регистр AX, а уже потом в переменную. Так что первая запись более эффективна и по коду, и по записи - так легче читается.

3. Утверждается, что лучшее сравнение - сравнение с 0. А здесь я вынужден согласиться. Дело в том, что сравнение с 0 осуществляется командой or (or si,si и т.п.), а другие сравнения - посредством команд cmp. Естественно, первое сравнение осуществляется быстрее. Поэтому, как утверждается, в циклических операторах параметр цикла лучше уменьшать.

Краткий справочник.

Более полное описание Вы сможете найти в руководствах по соответствующим языкам.

Передача параметров из функций в языках высокого уровня.

Турбо Си:

Структура длиной в 1 байт помещается в AL.

Структура длиной в 2 байта - в AX.

Четырехбайтовая структура - в DX:AX.

Числа типа FLOAT, DOUBLE и LONG через TOS (см. руководство по программированию на Си) или ST(0).

Структуры в 3 байта или более 5 байт - возвращается указатель на них: в AX для модели SMALL и в DX:AX для остальных моделей.

Турбо Паскаль:

Структура длиной в 1 байт помещается в AL.

Структура длиной в 2 байта - в AX.

Четырехбайтовая структура - в DX:AX.

Структура типа REAL передается в DX:BX:AX либо в ST(0), если используется сопроцессор.

Результат типа POINTER передается в DX:AX.

Для результатов функций типа STRING вызывающая программа помещает указатель на временную область памяти до помещения любых параметров, и функция возвращает строковое значение в эту временную память. Функция не должна удалять указатель.

Глава 16. Загружаемые драйверы.

Я вступил в темную аллею, под покров шумящих дубов, и с некоторым благоговением углублялся в мрак ее.

Н.М. Карамзин
Остров Борнгольм.

Файл IO.SYS содержит в себе драйверы стандартных устройств. Таких устройств как минимум 5. Мы уже говорили о них, однако, рассмотрим еще раз. Это экран, клавиатура, последовательный порт, стандартное устройство вывода сообщений об ошибках и принтер. Наряду со стандартными драйверами в MS DOS предусмотрена возможность, создавать свои загружаемые драйверы. Имя драйвера указывается в файле CONFIG.SYS после строки DEVICE=. Наверное, Вам приходилось сталкиваться с такими драйверами.

Загружаемые драйверы обладают рядом преимуществ по сравнению с обычными TSR-программами. Перечислю эти преимущества:

1. Загружаемые драйверы после загрузки становятся неотъемлемой частью операционной системы. Она выделяет им место в системной области и "знает" об их существовании.

2. Загружаемые драйверы попадают в память раньше, чем обычные программы и COMMAND.COM. Поэтому они раньше, чем другие программы, могут перехватить те или иные векторы прерываний, раньше выделить для себя память. Это может быть существенным, например, при борьбе с вирусами.

3. Интерфейс с загружаемыми драйверами можно осуществлять как посредством прерываний, так и посредством имени. Имя драйвера становится известным операционной системе после загрузки наряду с PRN, CON и т.д. Появляется возможность "научить" операционную систему работе с нестандартными устройствами.

4. Если имя устройства будет совпадать со стандартным именем, то Ваш драйвер заменит стандартный драйвер, содержащийся в IO.SYS. В примере, приведенном ниже, наш драйвер заменяет стандартный, обслуживающий устройство CON. Таким образом, весь вывод на экран, производимый с помощью функций DOS, будет идти только через наш драйвер.

Рассмотрим краткую теорию загружаемых драйверов. Более подробное описание Вы найдете в книгах [11, 23].

DOS управляет различными внешними устройствами, обращаясь к соответствующим драйверам. Каждый драйвер содержит имя устройства, которым он управляет. MS DOS находит нужный драйвер, просматривая список установленных драйверов, как стандартных, так и установленных в CONFIG.SYS. Первым в списке находится драйвер с именем NULL и содержит указатель на следующий драйвер. Точно так же другие драйверы содержат указатели на следующие. Указатель в последнем драйвере содержит -1, что означает конец списка. При установке нового драйвера он ставится сразу после NULL. Таким образом, стандартные драйверы DOS оказываются в списке

последними, и DOS обращается к ним только после просмотра всего списка, если в нем не найдено соответствующего имени. Возникает очень простой механизм замены стандартного драйвера на новый.

Драйвер должен состоять из трех частей:

1. Заголовок устройства. Помещается в начале драйвера и выглядит следующим образом.

DWORD Указатель на следующий драйвер в файле. Обычно -1, если драйвер последний или единственный.

WORD Атрибут.

WORD Указатель (смещение) на программу стратегии (см. ниже).

WORD Указатель (смещение) на программу прерывания (см. ниже).

8-BYTE Имя драйвера. Для того чтобы драйвер "откликался" на имя, длина его должна составлять восемь байт.

Существует два типа драйверов:

- драйвер символьного устройства,
- драйвер блокового устройства.

Символьное устройство осуществляет последовательный ввод-вывод. Такими устройствами являются консоль (CON), последовательный порт (AUX), принтер (PRN).

Блоковые устройства включают все дисководы в системе. Они могут осуществлять выборочный ввод/вывод блоков данных, обычно равных физическому размеру сектора. Все они имеют свои номера и идентифицируются буквами А, В, С и т.д. Один драйвер блокового устройства может отвечать за один или несколько логически связных дисководов. Предположим, данный драйвер отвечает за четыре дисковода. Это значит, что для него отведено четыре номера 0-3 и он занимает четыре буквы. Если он первый в списке драйверов блочных устройств, то эти буквы будут А, В, С, D. Следующему блочному драйверу будут отведены буквы, начиная с Е. Блочный драйвер не будет загружаться, если использован весь латинский алфавит (последняя буква Z). Драйверы символьных устройств могут отвечать только за одно устройство. Типичным примером драйвера блокового устройства является драйвер электронного диска.

В следующих двух таблицах разбирается структура атрибута устройства.

Символьное устройство.

Бит	Состояние	Значение
0	1	Консоль ввода
1	1	Консоль вывода
2	1	Нулевое устройство
3	1	Часы
4-5		Зарезервировано (нуль)
6	1	Поддерживает функции 3.2
7-10		Зарезервировано (нуль)

11	1	Понимает открыть/закрыть
12		Зарезервировано (нуль)
13	1	Поддерживает режим: вывод пока не занят
14	1	Поддерживает управляющие последовательности
15	1	Символьное устройство

Блоковые устройства.

Бит	Состояние	Значение
0		Зарезервировано (ноль)
1	1	Поддерживает 32-битную адресацию секторов
2-5		Зарезервировано (ноль)
6	1	Поддерживает функции 3.2 и функции общего ввода/вывода
7-10		Зарезервировано (нуль)
И	1	Понимает открыть/закрыть
12		Зарезервировано (ноль)
13	1	Определяет диск проверкой первого байта FAT
14	1	Поддерживает управляющие последовательности
15	0	Блоковое устройство

Посмотрите на Рис. 16.1, как заполнено поле атрибута.

2. Процедура стратегии. Служит для передачи драйверу команд и данных, а также для получения от драйвера кода возврата и данных. В процедуру стратегии через пару регистров ES:BX драйверу передается адрес запроса (см. ниже). В запросе содержится команда драйверу и необходимые для выполнения команды данные. Сюда же драйвер заносит код возврата и данные, передаваемые драйвером системе. Процедура стратегии должна быть оформлена как процедура типа FAR.

Процедура стратегии предусмотрена разработчиками операционной системы для дальнейшего использования в мультипрограммной среде. Процедура в нашей ситуации предельно проста - она передает ссылку на запрос системы для процедуры прерывания (см. Рис. 16.1).

Запрос системы состоит из двух частей: заголовок запроса, имеет жесткую структуру и сам запрос, формат которого зависит от команды, код которой содержится в заголовке. Мы разберем лишь заголовок запроса, любознательного же читателя я отсылаю к руководствам [5, 11], где подробно разбираются все виды запросов.

Заголовок запроса.

Длина	Значение
BYTE	Длина в байтах заголовка (число 13).
BYTE	Код устройства, к которому обращена операция (если драйвер обслуживает несколько устройств).
BYTE	Код операции (см. ниже).
WORD	Слово состояния. Заполняется процедурой прерывания.
8 BYTE	Зарезервировано.

3. Процедура прерывания анализирует передаваемый ей запрос, выполняет команду, код которой содержится в запросе, и выставляет в слове состояния результат выполненной операции. Фактически процедура прерывания является второй точкой входа. К ней DOS обращается уже после обращения к процедуре стратегии.

4. Кроме того, в теле драйвера могут храниться различные переменные (адрес заголовка запроса и др.), располагаться процедуры, к которым происходит обращение из процедуры прерывания.

Рассмотрим теперь все команды, выполняемые драйвером, и их краткую характеристику. Более подробные сведения можно найти в [11].

Команда инициализации (INI). Код 0. Выполняется всегда после загрузки драйвера (обязательно) и только один раз. Процедуру инициализации можно использовать для переустановки векторов прерываний или вывода сообщений. Должна возвращать конечный адрес резидентной части (по смещению 14) от начала заголовка запроса. Таким образом, сама процедура инициализации может быть исключена из резидентной части драйвера. Процедура инициализации - и только она - может использовать для работы функции DOS.

Команда проверки диска (CHECK_MEDIA). Код 1. Используется только с блоковыми устройствами. Проверяет, не менялся ли диск в дисководе.

Команда построения BPB (блок параметров BIOS) (BUILD_BPB). Код 2. Используется только с блоковыми устройствами. Вызывается каждый раз после сообщения о смене диска. Драйвер должен возвратить указатель на BPB.

УВВ чтение. Код 3 (IOCTL_INPUT). Эта команда используется драйвером для возвращения управляющей информации об устройстве (например, состояние принтера). Возвращается 44Н-Й функцией DOS и редко используется программами.

Чтение для блоковых и символьных устройств (INPUT). Код 4. Драйвер считывает данные с устройства и передает их операционной системе.

Процедура не буферизованного чтения (READ_NO_WAIT). Код операции - 5. Используется в драйверах символьных устройств. Фактически данные в операционную систему не передаются. DOS лишь получает сигнал о том, есть данные для ввода или нет.

Процедура проверки состояния (INPUT_STATUS). Код операции 6. Применима только для символьного устройства. От предыдущей операции она отличается тем, что проверяет не наличие символов для ввода, а готовность самого устройства.

Процедура сброса или очистки ввода (INPUT_FLUSH). Код операции 7. Применяется для символьных устройств, чтобы удалять из буфера ввода все вводимые ранее символы.

Запись для блоковых и символьных устройств (OUTPUT). Код операции 8. Предписывает драйверу произвести запись указанных данных на внешнее устройство.

Запись с проверкой (VERIFY_OUTPUT). Код 9. Команда аналогична предыдущей, но если установлен переключатель VERIFY ON, то после записи осуществляется считывание данных и их проверка. Разумеется, для принтеров и экранов такая команда не применима.

Команда состояния вывода (OUTPUT_STATUS). Код 10. Эта команда заставляет драйвер проверить состояние устройства, которое используется для вывода. Не применима для устройства, осуществляющего ввод.

Команда очистки вывода (OUTPUT_FLUSH). Код 11. Эта функция используется для сброса входной очереди на символьных устройствах. Драйвер осуществляет сброс, выставляет слово состояния и возвращает управление.

УВВ запись (IOCTL_OUTPUT). Код 12. Данные передаются драйверу, но не управляемому устройству. Разумеется, программа должна знать, какая информация может быть передана.

Процедуры закрытия и открытия (CLOSE и OPEN). Коды - 13 и 14. Процедуры вызываются DOS, если бит 11 в атрибуте заголовка установлен в 1. Процедуры предназначены для того, чтобы информировать драйвер о текущей активности файлов. Драйвер может вести счет открытым устройствам. В случае символьного устройства при вызове процедуры открытия драйвер может послать инициализирующую последовательность.

Процедура проверки типа диска (REMOVE_MEDIA). Код операции - 15. Функция вызывается если бит 11 атрибута заголовка драйвера равен 1. Некоторым утилитам иногда необходимо знать, с каким диском они работают (с флоппи или винчестером).

Вывод пока не занято (OUT_UN_BUSY). Код 16. Команда для символьного устройства. Действует, если установлен, бит 13 в слове атрибутов в заголовке устройства. Удобна для работы с принтерами.

Процедура общего УВВ. Код - 19. Функции общего УВВ дают расширенное средство управления вводом/выводом.

Процедуры установки/получения карты логического дисковода. Коды операций 23 и 24. Эти функции вызываются MS DOS, только если установлен бит 6 в слове атрибута заголовка устройства.

Блоковые устройства читают и записывают сектора, символьные устройства читают и записывают байты. По завершению ввода/вывода драйвер должен выставить слово-состояние и сообщить количество успешно переданных байт. В нижеприведенном примере мы не сообщаем счетчику, сколько байт выведено на экран, т.к. полагаем, что выведены все переданные байты.

Поле статуса.

Биты	Смысл поля
0-7	Коды ошибок. Работает только тогда, когда бит 15 равен 1.
8	Бит устанавливается в 1, когда драйвер закончил работу.
9	Устанавливается в 1, когда драйвер занят.
10-14	Резерв (нуль).
15	Устанавливается в 1, когда в работе драйвера произошла ошибка.

Коды ошибок драйвера.

Код	Вид ошибки
0	Попытка записи на защищенный от записи носитель.
1	Неизвестное устройство.
2	Устройство не готово.
3	Неизвестная команда.
4	Ошибка в контрольной сумме.
5	Плохая структура запроса драйвера.
6	Ошибка носителя.
7	Неизвестный носитель.
8	Сектор не найден.
9	В принтере нет бумаги.

Загружаемые драйверы строятся как COM-программы, однако ORG 100H в начале программы должно отсутствовать, так как при загрузке драйвера PSP не строится. Но при трансляции текста не обязательно доводить дело до COM-формата - EXE-формат так же будет понят системой (т.е. возможна, например, наряду с `device = drv.sys` и `device=drv.exe`).

На Рис. 16.1 представлен пример загружаемого драйвера. Вы можете оттранслировать его без всяких изменений и проверить на практике, как он работает. Данный драйвер заменит стандартный драйвер, обслуживающий устройство CON (т.е. по умолчанию вывод на дисплей). Причем драйвер будет перехватывать не только те вызовы, которые идут через команды COPY, TYPE или посредством открытия в языке высокого уровня файла с именем 'CON', но и, разумеется, символьный вывод посредством стандартных функций DOS.

Отметим некоторые особенности представленного драйвера, на которые следует обратить внимание.

1. В процедуре инициализации мы указываем адрес конца драйвера так, чтобы сама процедура в резидентную часть драйвера не вошла. Это вполне естественно, т.к. эта процедура исполняется всего один раз.

2. Вывод на экран осуществляется посредством прерывания ЮН (BIOS). Это достаточно очевидно, потому что вызов соответствующих функций DOS привел бы к вызову драйвером самого себя, не говоря уже о свойстве неинтерабельности DOS.

3. Весь вывод осуществляется посредством функции OUTPUT. Однако при символьном выводе система вызывает и другие функции. Для этих функций у нас стоит всего одна команда JMP QUIT - мы сообщаем системе, что данная функция выполнена успешно. Функции, где стоит JMP EXIT, будут возвращать в систему сообщение об ошибке.

4. В конце таблицы вызовов функций стоит 8 DUP(EXIT), т.е. восемь адресов EXIT. Дело в том, что функция OUT_UN_BUSY имеет номер 16, тогда как номер последней функции 24.

5. Мы не используем системный стек, а организуем свой. В целях безопасности рекомендуется поступать таким способом.

6. Наш драйвер будет работать только на вывод, команда DOS типа COPY CON A.TXT не будет работать. Для корректной работы драйвера следовало бы добавить процедуру ввода (INPUT).

7. Не будут работать и некоторые другие функции, традиционно обрабатываемые символьным выводом DOS. К таким, в частности, относится звуковой сигнал при выводе символа с кодом 7. Почему? Да просто мы это не предусмотрели. Поработайте над этим сами.

```
CODE  SEGMENT
DRIV  PROC  FAR
      ASSUME  CS:CODE
;заголовок драйвера  (устройства)
      DD        -1
ATRIBUT    DW           1010100000000010B
      DW        STRATEGY
      DW        INTERRUPT
      DB        'CON     '  ;имя, дополненное пробелами
                            ;до 8 байт
;процедура  стратегии
STRATEGY   PROC    FAR
      MOV       CS:_BX,BX
      MOV       CS:_ES,ES
      RETF
REQ        LABEL   DWORD
_BX        DW      ?
_ES        DW      ?
STRATEGY   ENDP
```

```
;здесь хранятся регистры
_SS     DW      ?
_SP     DW      ?
_AX     DW      ?
;наш стек
        DW      100 DUP(?)
STAC    DW      ?
;-----
;процедура прерывания
INTERRUPT  PROC    FAR
;устанавливаем свой стек
        MOV     CS:_AX,AX
        MOV     CS:_SS,SS
        MOV     CS:_SP,SP
        MOV     AX,CS
        MOV     SS,AX
        MOV     SP,OFFSET STAC
        MOV     AX,CS:_AX
;сохраняем все регистры
        PUSHF
        PUSH    DS
        PUSH    BX
        PUSH    SI
        PUSH    AX
        PUSH    DX
        PUSH    CX
        PUSH    DI
;определяем функцию драйвера
        LDS     SI,REQ
        XOR     BH,BH
        MOV     BL,DS:[SI+2]
        SHL     BX,1
        JMP     CS:[TABLE+BX]
;таблица переходов
TABLE      DW      INI
        DW      CHECK_MEDIA
        DW      BUILD_BPB
        DW      IOCTL_INPUT
        DW      INPUT
        DW      READ_NO_WAIT
        DW      INPUT_STATUS
        DW      INPUT_FLUSH
        DW      OUTPUT
        DW      VERIFY_OUTPUT
```

```
        DW          OUTPUT_STATUS
        DW          OUTPUT_FLUSH
        DW          IOCTL_OUTPUT
        DW          OPEN
        DW          CLOSE
        DW          REMOVE_MEDIA
        DW          OUT_UN_BUSY
        DW          8 DUP(EXIT)
;--
CHECK_MEDIA:            JMP EXIT
BUILD_BPB:              JMP EXIT
IOCTL_INPUT:            JMP QUIT
INPUT:                  JMP QUIT
READ_NO_WAIT:           JMP QUIT
INPUT_STATUS:           JMP QUIT
INPUT_FLUSH:            JMP QUIT
STROKA                  DB    'Нажмите любую клавишу'
NUM_STR                 DB    0              ;счетчик числа строк
_CX                     DW    ?
;процедура вывода текста из буфера запроса
OUTPUT:
        PUSH ES
;определяем, где курсор
        MOV AH,3
        MOV BH,0
        INT 10H
        CMP DH,24
        JNZ D6
        MOV AH,2
        MOV DH,23
        MOV DL,0
        INT ЮН
D6:
        MOV AX,DS
        MOV ES,AX
        LDS DI,ES:[SI+14] ;DS:DI на буфер
        MOV CX,ES:[SI+18] ;количество передаваемых байт
;цикл вывода текста
LOO:
        MOV CS:_CX,CX
;если конец строки, то следует перейти к следующей
        CMP DL,80
        JNZ  D4
        DEC  DI
```

```
        INC   CS:_CX
        JMP   SHORT  D5
D4:
        CMP BYTE PTR DS:[DI],13  ;признак конца
                                 ;строки?
        JNZ D1
D5:
        INC  CS:NUM_STR
        CMP DH,23
        JNZ D2
;сдвигаем экран вверх
        MOV AH,06
        XOR CX,CX
        MOV DH,23
        MOV DL,79
        MOV AL,1
        MOV BH,07H
        INT 10H
;курсор на начало строки 23
        MOV BH,0H
        MOV DH,23
        MOV DL,0
        MOV AH,2
        INT 10H
        JMP D3
D2:
;курсор на начало следующей строки
        MOV DL,0
        INC DH
        MOV AH,2
        INT 10H
        JMP D3
D1:
        MOV AL,DS:[DI]
        CMP AL,10      ;символ конца строки
                       ;не печатать
        JZ  D3
;печатаем символ
        MOV BH,0
        MOV CX,1
        MOV BL,7
        MOV AH,09H
        INT ЮН
```

```
;курсор к следующей позиции
        INC  DL
        MOV  AH,2
        INT  10H
D3:
;переходим к следующему символу
        INC  DI
        CMP  CS:NUM_STR,23
        JNZ  D7
        MOV  CS:NUM_STR,0
;выводим подсказку внизу экрана
        PUSH BX
        PUSH DI
        MOV  AX,0B800H
        MOV  ES,AX
        MOV  BX,3840
        MOV  DI,OFFSET STROKA
        MOV  CX,21
        MOV  AH,13
LOO1:
        MOV  AL,CS:[DI]
        MOV  ES: [BX],AL
        MOV  ES:[BX+1],AH
        INC  DI
        ADD  BX,2
        LOOP LOO1
;ждем нажатия клавиши
        MOV  AH, O
        INT  16H
        MOV  CX,21
        MOV  BX,3840
;чистим нижнюю строку экрана
D9:
        MOV  WORD PTR ES:[BX],0720H
        ADD  BX,2
        LOOP D9
        POP  DI
        POP  BX
D7:
        MOV  CX,CS:_CX
        DEC  CX
        JZ   D8
        JMP  LOO
```

```
D8:
      POP ES
      JMP QUIT
VERIFY_OUTPUT:        JMP QUIT
OUTPUT_STATUS:        JMP QUIT
OUTPUT_FLUSH:         JMP QUIT
IOCTL_OUTPUT:         JMP QUIT
OPEN:                 JMP QUIT
CLOSE:                JMP QUIT
REMOVE_MEDIA:         JMP EXIT
OUT_UN_BUSY:          JMP QUIT
EXIT:
      OR    WORD PTR DS: [SI+3], 8003H  ;установка
                                        ;битов ошибки
;8003H = 10000000B 03H - неизвестная команда.
QUIT:
      OR    WORD PTR DS: [SI+3],0100H   ;установка бита
                                        ;конец работы
;0100H =  00000001  00000000
      POP   DI
      POP   CX
      POP   DX
      POP   AX
      POP   SI
      POP   BX
      POP   DS
      POPF
      MOV   SS,CS:_SS
      MOV   SP,CS:_SP
      RETF
END_:
;функция инициализация
INI:
;устанавливаем в заголовке запроса адрес конца
;резидентной части драйвера
      MOV     WORD PTR DS: [SI+14],OFFSET END_
      MOV     DS: [SI+10H] , CS
      PUSH    CS
      POP     DS
;выводим сообщение о загрузке драйвера
      MOV     AH,9
      MOV     DX,OFFSET STR
      INT     21H
      JMP     QUIT
```

```
STR       DB    'Driver OUT_TEXT is set.',13,10,'$'
INTERRUPT    ENDP
DRIV         ENDP
CODE         ENDS
        END
```

Рис. 16.1. Пример простого драйвера, заменяющего устройство CON.

Глава 17. Работа с "мышью" на языке ассемблера.

"Mine is a long and a sad tale!" said the Mouse, turning to Alice, and sighing. "It is a long tail, certainly, " said Alice, looking down with wonder at the Mouse's tail; "but why do you call it sad?"

Alice's Adventures in Wonderland
by Lewis Carroll

Использование мыши совершило революцию в деле создания программного обеспечения. Теперь прикладные программы (для MS DOS), не поддерживающие это устройство, считаются уже вторым сортом. И вместе с тем имеется довольно много библиотек, включающих поддержку данного устройства. Прочтя данную главу, Вы сможете писать собственные библиотеки управления мышью. Автору пришлось написать несколько таких библиотек, и каждый раз процесс их создания доставлял ему массу удовольствия. Разумеется, читатель, речь у нас идет об операционной системе MS DOS, т.к. Windows сама берет на себя обработку функций мыши (см. главы 24, 25).

В качестве единицы перемещения мыши используется "микки". Это сигналы, которые подсчитывает мышь, а затем через определенные интервалы передает драйверу. Драйвер использует эти сигналы для определения положения мыши, преобразуя их в пиксели на экране. Драйвер автоматически перемещает курсор на экране в соответствие с движением мыши и отслеживает координаты курсора, предоставляет пользовательской программе ряд функций, позволяющих использовать мышь в программе. Основой взаимодействия программы с драйвером является возможность установки процедуры прерывания по одному из следующих событий:

1. Нажатие или отпускание кнопки мыши.
2. Передвижение мыши.

Несмотря на то что включение справочника по основным функциям драйвера мыши сильно увеличило размер данной главы, я все-таки сделал это. Поэтому данная глава является достаточно полным пособием по созданию библиотеки поддержки мыши. Ниже представлен полный перечень базовых функций драйвера мыши фирмы Микрософт. Другие драйверы также поддерживают данный набор функций, но могут иметь и другие возможности.

I.

Функция 0: начальная установка драйвера и чтение текущего состояния. Функция О производит начальную установку и возвращает информацию о текущем состоянии аппаратных и программных средств мыши. Функция определяет текущий режим экрана, прячет курсор и помещает его в центр экрана, а также задает начальные значения внутренним переменным драйвера в соответствии со следующей таблицей:

Переменная	Значение
внутренний флажок курсора	OFFFFH(-l), курсор спрятан
форма графического курсора	горизонтальный овал
текстовый курсор	обращенное видеоизображение символа
маски, определяемые пользователем	все нули
режим эмуляции	включен
вертикальное соотношение микки/пиксель	16/8
горизонтальное соотношение микки/пиксель	8/8
минимальные координаты курсора по вертикали и горизонтали максимальные координаты курсора по вертикали и горизонтали	0,0 соответствующие максимальные значения координат экрана в текущем режиме, минус единица

Входные значения	AX	0000H
Возвращаемые значения	AX	0000H драйвер или аппаратная поддержка не установлены,
	FFFFH(-1) - начальная установка прошла успешно	
	BX	количество кнопок "мыши"
	OOOOH	не две кнопки
	0002H	две кнопки
	OOO3H	три кнопки

Функция 1: Сделать курсор видимым. Функция 1 увеличивает на 1 значение счетчика внутреннего флажка курсора. Если флажок равен 0, курсор становится видимым и появляется на экране. Начальное значение флажка равно -1, что соответствует спрятанному курсору. В случае, если внутренний флажок уже равен 0, вызов функции 1 не приводит ни к каким результатам.

Входные значения	AX	0001H
Возвращаемые значения	нет	

Функция 2: Сделать курсор невидимым. Функция 2 делает курсор невидимым и уменьшает на 1 значение внутреннего флажка курсора. Несмотря на то что курсор не виден на экране, он продолжает отслеживать движение "мыши".

Эту функцию рекомендуется вызывать каждый раз перед тем, как произвести какие-либо изменения на том участке экрана, где находится курсор. Этим Вы избежите проблем, возникающих из-за взаимодействия курсора с данными на экране.

Нужно иметь в виду, что на каждый вызов функции 2 впоследствии должен быть произведен вызов функции 1, для того чтобы восстановить внутреннее значение флажка курсора. Кроме того, при каждом переключении режима экрана функция 2 вызывается автоматически.

Произведите вызов функции 2 в конце Вашей программы, чтобы спрятать курсор. Это позволит быть уверенным в том, что ничего лишнего не останется на экране после завершения программы.

Входные значения	AX	0002H
Возвращаемые значения	нет	

Функция 3: Определяет местоположение курсора и состояние кнопок мыши. Функция 3 возвращает данные по текущему состоянию кнопок мыши и положения курсора на экране.

Входные значения	AX	0003H
Возвращаемые значения	BX	байт состояния кнопок (BL)
	Бит 0	левая кнопка
	Бит 1	правая кнопка
	Бит 2	средняя кнопка (мышь MOUSE SYSTEMS)
	Биты 3-7	не используются
	CX	горизонтальная координата курсора
	DX	вертикальная координата курсора

Если бит установлен (1), то кнопка нажата, если сброшен (0) - кнопка отпущена.

Функция 4: Установить курсор на экране в заданную позицию. Функция устанавливает курсор на экране в заданную позицию. Входные значения координат должны быть в пределах диапазона, установленного драйвером для текущего виртуального экрана мыши.

Некоторые драйверы мыши округляют значение счетчика микки для получения действительной координаты пикселя, другие просто отбрасывают остаток при масштабировании. Это может приводить к некоторой разнице в позиционировании курсора при использовании разных драйверов мыши, что становится особенно заметным при сочетании новых мышей высокого разрешения с дисплеями высокого разрешения.

Входные значения	AX	0004H
	CX	горизонтальная координата курсора
	DX	вертикальная координата курсора
Возвращаемые значения	нет	

Режим экрана	Адаптер дисплея	Виртуальный экран (XXY)	Размер цели	Бит/Пиксель Графический режим
0	C, E, 3270	640 X 200	16 X 8	-
1	C, E, 3270	640 X 200	16X8	-
2	C, E, 3270	640 X 200	8 X 8	-
3	C, E, 3270	640 X 200	8 X 8	-
4	C, E, 3270	640 X 200	2 X 1	2
5	C, E, 3270	640 X 200	2 X 1	2
6	C, E, 3270	640 X 200	1 X 1	1
7	M, E, 3270	640 X 200	8 X 8	-
B	E	640 X 200	16X8	2
E	E	640X200	1X1	1
F	E	640X350	1X1	1
10	E	640X350	1X1	1
30	3270	720X350	1X1	1
	H	720X348	1X1	1

Адаптер дисплея:
M= IBM Монохромный дисплей/принтер адаптер
C = IBMCGA
E= IBM EGA
3270 = IBM 3270 PC
H = Геркулес монохромная графическая карта

Функция 5: Получить информацию о количестве нажатий на одну кнопку. Функция 5 возвращает информацию о текущем состоянии кнопок, количестве нажатий на указанную кнопку с момента последнего вызова данной функции, а также о позиции курсора в момент последнего нажатия указанной кнопки.

Диапазон счетчика нажатий колеблется от 0 до 0EFFFH. Индикатор переполнения отсутствует. После вызова данной функции счетчик обнуляется.

Входные значения	AX	0005H
	BX	байт идентификации кнопок (BL)
	Бит 0	левая кнопка
	Бит 1	правая кнопка
	Бит 2	средняя кнопка
Возвращаемые значения	AX	байт состояния кнопок (AL)
		Бит 0 левая кнопка

		Бит 1 правая кнопка
		Бит 2 средняя кнопка
		(мышь MOUSE SYSTEMS)
	BX	число нажатий указанной кнопки с момента последнего вызова функции
	CX	горизонтальная координата курсора в момент нажатия указанной кнопки
	DX	вертикальная координата курсора в момент нажатия указанной кнопки.

Если бит установлен (1), то кнопка нажата, если сброшен (0) - кнопка отпущена.

Функция 6: Получить информацию о количестве отпусканий кнопки. Функция 6 аналогично функции 5 возвращает информацию о текущем состоянии кнопок, количестве отпусканий указанной кнопки с момента последнего вызова данной функции, а также о позиции курсора в момент последнего отпускания указанной кнопки.

Диапазон счетчика отпусканий колеблется от 0 до OEFFFH. Индикатор переполнения отсутствует. После вызова данной функции счетчик обнуляется.

Входные значения	AX	0006H
	BX	байт идентификации кнопок (BL)
	Бит 0	левая кнопка
	Бит 1	правая кнопка
	Бит 2	средняя кнопка
Возвращаемые значения	AX	байт состояния **кнопок** (AL)
		Бит 0 левая кнопка
		Бит 1 правая кнопка
		Бит 2 средняя кнопка
	BX	число отпусканий указанной кнопки с момента последнего вызова функции
	CX	горизонтальная координата курсора в момент отпускания указанной кнопки
	DX	вертикальная координата курсора в момент отпускания указанной кнопки.

Если бит, установлен (1), то кнопка нажата, если сброшен (0) - кнопка отпущена.

Функция 7: Установить диапазон перемещения курсора по горизонтали (X). Функция 7 устанавливает горизонтальный диапазон перемещения курсора на экране. В результате текущая горизонтальная координата курсора приводит к новому масштабу. Если в момент вызова горизонтальная координата курсора находилась вне указанного диапазона, курсор помещается в соответствующий край диапазона.

Входные значения	AX	0007H
	OC	минимальная горизонтальная координата курсора
	DX	максимальная горизонтальная координата курсора
Возвращаемые значения	нет	

Если минимальное значение больше максимального, функция производит обмен значений.

Функция 8: Установить диапазон перемещения курсора по вертикали (Y). Функция 8 устанавливает вертикальный диапазон перемещения курсора на экране. В результате текущая вертикальная координата курсора приводится к новому масштабу. Если в момент вызова вертикальная координата курсора находилась вне указанного диапазона, курсор помещается в соответствующий край диапазона.

Входные значения	AX	0008H
	CX	минимальная вертикальная координата курсора
	DX	максимальная вертикальная координата курсора
Возвращаемые значения	нет	

Если минимальное значение больше максимального, функция производит обмен значений.

Функция 9: Задает параметры графического курсора. Функция 9 устанавливает цвет, форму и задает координаты "горячего пятна" графического курсора. Как уже было сказано выше (см. Примечание ф.9), этот курсор формируется из графического блока размером 16 на 16 пикселей и определяется двумя массивами 16 на 16 бит каждый (маской экрана и маской курсора). Координаты горячего пятна курсора должны находиться в диапазоне от 0 до 16.

Входные значения	AX	0009H
	BX	горизонтальная координата горячего пятна курсора в маске курсора
	DX	вертикальная координата горячего пятна курсора в маске курсора
	ES:DX	указатель на массив масок
	16 слов - маска экрана	
	16 слов - маска курсора	
Возвращаемые значения	нет	

Каждое слово задается значениями 16 пикселей в соответствующем ряду. Младший бит соответствует крайнему правому пикселю.

Функция 10: Задать параметры текстового курсора. Функция 10 осуществляет выбор жесткого или мягкого текстового курсора. Для жесткого текстового курсора задаются первая и последняя скан-линии, принимающие участие в формировании курсора. Для мягкого текстового курсора задаются маски экрана и курсора.

Входные значения	AX	OOOAH
	BX выбор типа курсора	
	OOH мягкий текстовый курсор	
	01H жесткий текстовый курсор	
	CX маска экрана или номер первой скан-линии	
	DX маска курсора или номер последней скан-линии	
Возвращаемые значения	нет	

Функция 11: Прочитать значение счетчика сигналов микки. Функция 11 возвращает число сигналов микки (минимальных приращений перемещения мыши, регистрируемых аппаратными средствами), накопленное счетчиком с момента вызова данной функции. Положительные значения соответствуют движению мыши вправо и вверх. Значения счетчика расположены в интервале от -32768 до 32767. Индикатор переполнения отсутствует. После вызова функции микки обнуляются.

Входные значения	AX	OOOBH
Возвращаемые значения	CX	число микки по горизонтали
	DX	число микки по вертикали

Функция 12: Задает адрес подпрограммы обработки прерываний. Функция 12 передает драйверу адрес входа в подпрограмму обработки прерывания, вызванного некоторым событием. В случае возникновения такого события выполнение прикладной программы временно прерывается, и управление передается по установленному адресу. После завершения работы подпрограммы работа прикладной программы возобновляется в той точке, где была прервана.

Битовая маска определяет условия, вызывающие прерывание. Установленный бит (1) разрешает прерывание, сброшенный (0) - запрещает.

Входные значения	AX	OOOCH
	CX	маска вызова
	бит 0	изменение позиции курсора
	бит 1	нажата левая кнопка
	бит 2	отпущена левая кнопка

бит 3	нажата правая кнопка
бит 4	отпущена правая кнопка
бит 5	нажата средняя кнопка (MOUSE SYSTEMS)
бит 6	отпущена средняя кнопка (MOUSE SYSTEMS)
7 - 15	не используются
ES:DX FAR	адрес подпрограммы обработки прерываний

Возвращаемые значения не определены

Драйвер фирмы MISROSOFT следит за позицией мыши. Драйверы фирм LOGITECH и MOUSE SYSTEMS следят за позицией курсора.

Во время вызова подпрограммы обработки прерываний драйвер передает ей следующие параметры:

AH	маска условия (биты установлены так же, как и в маске вызова)
BX	состояние кнопок
CX	горизонтальная координата курсора
DX	вертикальная координата курсора
ЕЯ	значение вертикального счетчика микки
SI	значение горизонтального счетчика микки

При завершении работы вашей прикладной программы установите все биты маски вызова равными 0 и вызовите функцию 12. (При выходе из программы оставляйте систему в том же состоянии, что и при входе.)

Функция 13: Включение режима эмуляции светового пера. Функция 13 позволяет работать мыши в режиме светового пера. В этом случае обращение к функции светового пера будет возвращать координаты курсора на момент последнего состояния "перо опущено".

Состояния "перо опущено" и "перо поднято" определяются кнопками мыши: все кнопки отпущены - "перо поднято", одна кнопка нажата - "перо опущено".

При начальной установке драйвера режим эмуляции светового пера включается по умолчанию. Если в системе присутствует настоящее световое перо, должна быть вызвана функция 14 для запрещения эмуляции.

Входные значения	AX	000DH
Возвращаемые значения	нет	

Функция 14: Запрещение режима эмуляции светового пера. Функция 14 запрещение режим эмуляции светового пера. При этом любое последующее обращение к фун-

кции светового пера будет возвращать информацию только о состоянии настоящего светового пера.

Входные значения	AX	000EH
Возвращаемые значения	нет	

Функция 15: Установить соотношение микки/пикселей. Соотношение определяется числом **микки,** приходящимся на 8 пикселей экрана. Значения микки должны быть в пределах от 1 до OEFFFH. При начальной установке драйвера по умолчанию выбираются следующие соотношения микки/пикселей: по горизонтали - 8/8; по вертикали - 16/8.

Входные значения	AX	000FH
	CX	число микки на 8 пикселей по горизонтали
	DX	число микки на 8 пикселей по вертикали
Возвращаемые значения	нет	

Функция 16: Запретить появление курсора в специальной области. Функция 16 создает на экране условную специальную область. Если курсор попадает в эту область, то он исчезает. Для отмены действия специальной области необходимо произвести вызов функции 1.

Входные значения	AX	00ЮН
	ES:DX	указатель на **массив,** определяющий специальную область
Возвращаемые значения	нет	

Формат массива:

Смещение	Параметр
OOOOH	левая горизонтальная координата
0002H	верхняя вертикальная координата
0004H	правая горизонтальная координата
0006H	нижняя вертикальная координата

Функция 17: Задать параметры большого блока графического курсора.

Примечание:

Функция определена для драйвера PC MOUSE. Фирма MICROSOFT **не** дает документации по этой функции.

Действие функции 17 аналогично действию функции 9. Отличие состоит в том, что функция 17 задается размером массивов, определяющих маски экрана и курсора.

Входные значения	AX	0011H
	BH	ширина курсора в словах (2 байта)
	CH	количество рядов в курсоре по вертикали

	BL	горизонтальная координата горячего пятна курсора в маске курсора (от -16 до 16)
	ES:DX	указатель на массивы масок
	(BH 'CH)	слов - маска экрана
	(BH 'CH)	слов - маска курсора
Возвращаемые значения	AH	OFFFFH (-1) функция выполнена успешно

Функция 18: НЕ используется.

Функция 19: Установить порог удвоенной скорости. Функция устанавливает значение порога удвоенной скорости движения мыши (измеряемого как микки в секунду), при превышении которого скорость движения курсора на экране удваивается. По умолчанию величина порога удвоенной скорости составляет 64 микки в секунду.

Входные значения	AX	0013H
	DX	порог скорости в микки в секунду
Возвращаемые значения	нет	

Функция 20: Установить временную подпрограмму обработки прерывания. Функция 20 позволяет вам временно установить подпрограмму обработки прерываний, поступающих от мыши, либо просто изменить маску вызова (см. ф. 12). Адрес входа в старую подпрограмму должен быть обязательно восстановлен перед тем, как ваша прикладная программа закончила работу.

Входные значения	AX	0014H
	BX:DX FAR	указатель на новую программу
	CX	новая маска вызова (см. ф.12)
		бит 0 изменение позиции курсора
		бит 1 нажата левая кнопка
		бит 2 отпущена левая кнопка
		бит 3 нажата правая кнопка
		бит 4 отпущена правая кнопка.
		бит 5 нажата средняя кнопка (MOUSE SYSTEMS)
		бит 6 отпущена средняя кнопка (MOUSE SYSTEMS)
		7 - 15 не используются
Возвращаемые значения	BX:DX	FAR адрес старой подпрограммы обработки прерываний
	CX	маска вызова старой подпрограммы

Эта функция в своей работе использует регистры AX,BX,CX,DX,SI,DI,DS и ES.

Функция 21: Получить данные о размере буфера для записи состояния драйвера. Функция 21 возвращает данные о размере необходимого буфера, в который будут записаны параметры текущего состояния драйвера мыши, с целью их последующего восстановления.

Входные значения	AX	0015H
Возвращаемые значения	BX	размер буфера, необходимого для записи текущего состояния драйвера (байт)

Функция 22: Записать параметры текущего состояния драйвера в буфер. Функция 22 записывает параметры текущего состояния драйвера мыши в буфер с целью их последующего восстановления. Размер необходимого буфера определяется с помощью функции 21.

Входные значения	AX	0016H
	ES:DX	указатель в буфер
Возвращаемые значения	нет	

Функция 23: восстановить параметры состояния драйвера из буфера. Функция 23 позволяет восстановить параметры предыдущего состояния драйвера мыши, записанные в буфер (см. функции 21, 22).

Входные значения	AX	0017H
	ES:DX	указатель на буфер
Возвращаемые значения	нет	

Функция 24: Установить альтернативную подпрограмму обработки прерываний. Функция работает аналогично функции 12. Она позволяет установить альтернативную подпрограмму обработки тех прерываний мыши, которые не были включены в подпрограмму, установленную функцией 12. Вы можете установить до трех различных подпрограмм обработки прерываний путем последовательных обращений к функции 24.

Входные значения	AX	0018H
	CX	маска вызова бит 0 изменение позиции курсора бит 1 нажата левая кнопка бит 2 отпущена левая кнопка бит 3 нажата правая кнопка бит 4 отпущена правая кнопка бит 5 клавиша SHIFT была нажата в момент прерывания бит 6 клавиша CTRL была нажата в момент прерывания

		бит 7 клавиша ALT была нажата в момент прерывания
		8 - 15 не используются
	ES:DX FAR	Указатель на подпрограммы обработки прерываний
Возвращаемые значения	AX	0FFFFH - ошибка

Установленный бит (1) разрешает прерывание, сброшенный бит (0) запрещает. Во время вызова подпрограммы обработки прерывания драйвер передает ей следующие параметры:

AH	маска условия (биты установлены так же, как и в маске вызова)
BX	состояние кнопок
CX	горизонтальная координата курсора
DX	вертикальная координата курсора
DI	значение вертикального счетчика микки
SI	значение горизонтального счетчика микки

Функция 25: Получить вектор прерывания, указывающий на установленную пользователем подпрограмму обработки. Функция 25 осуществляет попытку найти установленную пользователем подпрограмму обработки прерывания (определенную функцией 12), маска вызова которой соответствует маске, содержащей в CX.

Входные значения	AX	0019H
		CX маска вызова (аналогично функции 24)
Возвращаемые значения	AX	OFFFFH вектор или маска не найдены
	BX:DX	указатель на вектор прерывания пользователя (0 если AX = OFFFFH)
	CX	маска вызова (0 если AX = OFFFFH)

Функция 26: Установить порог чувствительности мыши. Функция 26 позволяет установить порог чувствительности мыши по горизонтальной скорости движения, вертикальной скорости движения, а также порог ускоренной скорости мыши.

Входные значения	AX	001AH
	BX	горизонтальная скорость
	CX	вертикальная скорость
	DX	порог удвоенной скорости микки в секунду
		0000H устанавливает стандартный порог 64/сек
Возвращаемые значения	нет	

Функция 27: Получить данные о чувствительности мыши. Функция 27 возвращает параметры, установленные функцией 26.

Входные значения	AX	001BH
Возвращаемые значения	BX	горизонтальная скорость
	CX	вертикальная скорость микки в секунду
		0000H порог 64/сек

Функция 28: Установить частоту прерываний мыши. Функция 28 позволяет установить частоту прерываний, вырабатываемой аппаратными средствами мыши для передачи драйверу информации о своем состоянии и накоплении числа сигналов микки.

Входные значения	AX	001CH
	BX	желаемая частота прерываний (BL)

OOH	прерывания запрещены
01H	30 прерываний в секунду
02H	50 прерываний в секунду
O3H	100 прерываний в секунду
04H	200 прерываний в секунду
05H-FFH	не определено

Возвращаемые значения нет

Если функция передает значение большее 04H (BL), драйвер аппаратной поддержки MICROSOFT INPORT может вести себя непредсказуемо.

Функция 29: Установить страницу дисплея. Функция 29 переключает экран на указанную страницу (курсор при этом становится видимым).

Входные значения	AX	001DH
	BX	номер страницы дисплея (0 - 7)
Возвращаемые значения	нет	

Функция 30: Получить номер текущей страницы дисплея.

Входные значения	AX	001EH
Возвращаемые значения	BX	номер текущей страницы дисплея (0 - 7)

Функция 31: Запретить работу драйвера мыши. Функция 31 запрещает работу драйвера мыши, восстанавливая значения векторов прерываний 33H, ЮН и 71H (для процессора 8086) или 74H (для процессора 80286/386). Функция возвращает значение старого вектора прерываний 33H, указывающего либо на интерфейс мыши более высокого уровня BX:CX, либо первоначальное значение этого вектора ES:BX. В последнем случае работа драйвера мыши будет полностью запрещена.

Входные значения	AX	001FH
Возвращаемые значения	AX	001FH функция выполнена успешно 0FFFFH ошибка
	BX:CX	старый вектор прерывания 33H, указывающий на интерфейс мыши более высокого уровня
	ES:BX	первоначальное значение вектора прерывания 33H

Функция 32: восстановить работу драйвера мыши. Функция 32 восстанавливает прежнее значение векторов прерываний ЮН и 71H (8086) или 74H (80286/386), измененные функцией 31.

Входные значения	AX	0020H
Возвращаемые значения	нет	

Функция 33: Произвести начальную установку программного обеспечения мыши. Действие функции 33 аналогично действию функции 0 с той разницей, что функция 33 не производит начальной установки аппаратных средств мыши.

Входные значения	AX	0021H
Возвращаемые значения	AX	0021H драйвер мыши не установлен OFFFFH драйвер мыши установлен
	BX	0002H начальная установка произведена

Функция 34: Выбрать язык для выдачи диагностических сообщений.

Входные значения	AX	0022H
	BX	код языка (BL) OOH Английский 01H Французский 02H Голландский O3H Немецкий 04H Шведский 05H Финский 06H Испанский 07H Португальский 08H Итальянский другие значения не используются
Возвращаемые значения	нет	

Значения, отличные от OOH, могут быть использованы только для драйвера MICROSOFT INTERNATIONAL MOUSE.

Функция 35: Получить информацию об используемом языке.

Входные значения	AX	0023H
Возвращаемые значения	BX	код языка (BL) OOH Английский 01H Французский 02H Голландский O3H Немецкий 04H Шведский 05H Финский 06H Испанский 07H Португальский 08H Итальянский другие значения не используются

Функция 36: Получить дополнительную информацию о мыши.

Входные значения	AX	0024H
Возвращаемые значения	AX	0FFFH ошибка, иначе BH:BL номер версии драйвера
	CH	тип интерфейса мыши 01H мышь, подключаемая к общей шине 02H мышь, подключаемая к последовательному порту O3H мышь, подключаемая к MICROSOFT INPORT 04H мышь, подключаемая к IBM PS/2 roiNnNGDEVICEPORT 05H мышь HEWELETT-PACKARD
	CL	номер запроса прерывания (IRQ) OOH PS/2 позиционирующее устройство 01H не определено 02HIRQ2 03H IRQ3 ... ... 07HIRQ7

П.

Выше был представлен справочник функций драйвера мыши. Опираясь на этот справочник, Вы сможете создавать свои библиотеки поддержки мыши как для ассемблера, так и для языков высокого уровня. Данная информация в несколько переработанном виде взята автором из [5, 13, 24].

Ниже представлена программа, демонстрирующая курсоры мыши в различных режимах экрана. Кроме курсора, выводится еще графическая точку (в текстовых режимах, разумеется, она не появляется), чтобы определить наличие графического режима.

```
CODE   SEGMENT
       ASSUME CS:CODE, DS:CODE
       ORG   100H
BEGIN:
       MOV   BYTE PTR SCR,0
       CALL  SET_MOUS
CONT:
       CALL  SCREEN
       CALL  CUR_VIS
       CALL  POINT
       CALL  KEY
       INC   BYTE PTR SCR
       CMP   BYTE PTR SCR,20
       JNZ   CONT
       MOV   BYTE PTR SCR,2
       CALL  SCREEN
       CALL  CUR_NO_VIS
       RET
;начальная установка мыши
SET_MOUS  PROC
       MOV AX,0
       INT 33H
       RET
SET_MOUS  ENDP
;курсор мыши видимый
CUR_VIS  PROC
       MOV AX,1
       INT 33H
       RET
CUR_VIS  ENDP
;курсор мыши невидимый
CUR_NO_VIS  PROC
       MOV AX,2
       INT 33H
       RET
```

```
CUR_NO_VIS ENDP
;режим экрана
SCREEN  PROC
        XOR AH,AH
        MOV AL,SCR
        INT 10H
        RET
SCREEN  ENDP
; поставить точку
POINT  PROC
        MOV AH,0CH
        XOR BH,BH
        MOV DX,20
        MOV CX,40
        MOV AL,1
        INT 10H
        RET
POINT ENDP
;ждать нажатие  клавиши
KEY  PROC
        XOR AH,AH
        INT 16H
        RET
KEY ENDP
SCR   DB  0  ;хранится текущий режим экрана
CODE ENDS
        END BEGIN
```

Рис. 17.1. Пример стандартных курсоров мыши в различных режимах экрана.

Таким образом, драйвер мыши поддерживает курсор мыши как в текстовом, так и графическом режиме. Если данный курсор почему-либо не удовлетворяет Вас, с помощью функций 9 и 10 Вы можете создать свой курсор как для текстового, так и для графического режима.

Рассмотрим вначале формирование графического курсора (функция 9). Графический курсор определяются прямоугольным блоком пикселей. Форма графического курсора и его взаимодействие с экраном определяется содержимым двух массивов. Размер каждого массива — 16 на 16 бит. Первый массив называется маской экрана, второй - маской курсора. Маска экрана определяет, какие пиксели будут участвовать в формировании образа курсора, а какая часть будет формировать фон. Маска курсора определяет те пиксели, которые участвуют в формировании цвета курсора. Механизм формирования графического курсора следующий: для каждого пикселя части экрана под курсором производится побитовая операция AND с маской экрана. Смысл здесь следующий: если бит маски равен 0, то соответствующий пиксель получает нулевой атрибут, т.е. черный цвет, если же бит маски равен 1, то соответствующий пиксель остается, каким был. Пос-

ле выполнения этой операции над ее результатом проводится побитовая операция XOR с маской курсора. Вдумавшись, можно сообразить, что соль этих операций заключается в том, будет или нет курсор мыши или его части прозрачными или не нет. Предположим, нужно, чтобы через данную точку курсора всегда просвечивала точка экрана, в этом случае в маске экрана будет стоять 1, а в маске курсора 0.

При формировании текстового курсора (функция 10, 0 в ВХ) используются две шестнадцатибитные маски: маска экрана и маска курсора. Структура их такова: 15-й бит - мигающий (1) или немигающий (0) курсор, 12-14 биты - цвет фона данной текстовой клетки, 8-11 биты - цвет символа, 0-7 биты - код символа. Механизм установки текстового курсора таков: драйвер вначале производит операцию AND над содержимым клетки и маской экрана, затем операцию XOR над результатом и маской курсора. Например, Вы хотите, чтобы курсор мыши был просвечивающим прямоугольником. В этом случае код символа в маске экрана будет 255 (11111111B), а код символа в маске курсора будет 0. И наоборот, если Вы хотите получить непросвечивающий курсор, то код символа в маске экрана должен быть 0. Код же символа в маске курсора даст символ, который будет изображать курсор мыши. Отсюда, кстати, следует, что, если необходимо, чтобы в текстовом режиме курсор мыши имел нестандартную форму, следует создать собственный символ (см. главу 7).

```
CODE SEGMENT
      ASSUME CS:CODE, DS:CODE
      ORG 100H
BEGIN:
      MOV  BYTE PTR SCR,0
      CALL SET_MOUS
;параметры графического курсора
      MOV  AX,9
      MOV  BX,20
      MOV  CX,10
      LEA  DX,M1
      INT  33H
;параметры мягкого текстового курсора
      MOV  AX,0AH
      MOV  BX,0   ;мягкий текстовый курсор
;курсор в виде просвечивающего прямоугольника
      MOV  CX,0111111111111111B      /маска экрана
      MOV  DX,0001001000000000B      ;маска курсора
      INT  33H
CONT:
      CALL SCREEN
      CALL CUR_VIS
      CALL POINT
      CALL KEY
      INC  BYTE PTR SCR
      CMP  BYTE PTR SCR,20
```

```
        JNZ   CONT
        MOV   BYTE PTR SCR,2
        CALL  SCREEN
        CALL  CUR_NO_VIS
        RET
;начальная установка мыши
SET_MOUS PROC
        MOV AX,0
        INT 33H
        RET
SET_MOUS ENDP
;курсор мыши видимый
CUR_VIS PROC
        MOV AX,1
        INT 33H
        RET
CUR_VIS ENDP
;курсор мыши невидимый
CUR_NO_VIS PROC
        MOV AX,2
        INT 33H
        RET
CUR_NO_VIS ENDP
;режим экрана
SCREEN PROC
        XOR AH,AH
        MOV AL,SCR
        INT 10H
        RET
SCREEN ENDP
;поставить точку
POINT PROC
        MOV AH,0CH
        XOR BH,BH
        MOV DX,20
        MOV CX,40
        MOV AL,1
        INT 10H
        RET
POINT ENDP
;ждать нажатие клавиши
KEY PROC
        XOR AH,AH
        INT 16H
        RET
```

```
KEY ENDP
SCR  DB 0 ;хранится текущий режим экрана
M1:
      DW 0000000000000000B
      DW 0111111111111110B
      DW 0111111111111110B
      DW 0111111111111110B
      DW 0111111111111110B
      DW 0111111111111110B
      DW 0111111111111110B
      DW 0111111111111110B
      DW 0111111111111110B
      DW 0111111111111110B
      DW 0111111111111110B
      DW 0111111111111110B
      DW 0111111111111110B
      DW 0111111111111110B
      DW 0111111111111110B
      DW 0000000000000000B
M2:
      DW 1111111111111111B
      DW 1000000000000001B
      DW 1000000000000001B
      DW 1000000000000001B
      DW 1000000000000001B
      DW 1000000000000001B
      DW 1000000000000001B
      DW 1000000000000001B
      DW 1000000000000001B
      DW 1000000000000001B
      DW 1000000000000001B
      DW 1000000000000001B
      DW 1000000000000001B
      DW 1000000000000001B
      DW 1000000000000001B
      DW 1111111111111111B
CODE ENDS
      END BEGIN
```

Рис. 17.2. Программа, формирующая графический и текстовый курсор.

Программа, представленная на Рис. 17.2, аналогична программе на Рис. 17.1 с той лишь разницей, что и графический, и текстовый курсоры формируются прямо в программе.

Из сказанного выше ясно, что на курсор мыши в текстовом режиме накладываются жесткие ограничения. Например, он может двигаться только от знака к знаку, т.е. с

шагом, равным нескольким графическим точкам. Чтобы сделать движение курсора в текстовом режиме плавным, придется создавать курсор, состоящий из нескольких символов, т.е. программировать знакогенератор.

III.

Основная идея использования мыши в программе - это отслеживание положения курсора мыши и адекватное на это реагирование. Можно было бы непрерывно проверять положение мыши, а также проверять, нажата или не нажата та или иная кнопка. Но этот дилетантский подход хотя и возможен, но на практике никем не применяется. Взаимодействие программы с мышью происходит по прерыванию. Возможность установить процедуру прерывания (и даже не одну) по какому-либо событию предоставляет драйвер мыши (см. функции 12 и 20).

Пусть процедура обработки прерывания от мыши называется INT_MOUS. Тогда фрагмент, устанавливающий эту процедуру в активное состояние, будет таким:

```
MOV AX,0CH
MOV BX,SEG INT_MOUS
MOV ES,BX                    ;сегментный адрес
LEA DX,INT_MOUS              ;смещение
MOV CX,0000000000001011B     ;маска
INT 33H
```

Процедура прерывания будет вызываться при движении мыши, а также при нажатии левой и правой кнопки. Для того чтобы дезактивировать процедуру прерывания, следует выполнить такой же фрагмент, только в CX должен находиться 0.

Итак, с установкой процедуры прерывания все ясно. Вопрос второй - как процедура прерывания будет взаимодействовать с основной программой. Здесь возможно два пути:

1. Процедура прерывания делает что-то сама при наступлении какого-либо события. Такой подход желателен, если Вы хотите, чтобы действие мыши давало в любой ситуации один и тот же результат. Например, появление курсора мыши в правом верхнем углу экрана вызовет возникновение "звездного неба". Здесь, по сути, нет никакого взаимодействия с основной программой, все происходит помимо нее.

2. Взаимодействие осуществляется посредством буфера клавиатуры. Действие мыши вызывает имитацию нажатия каких-либо реально существующих или фиктивных клавиш. Имитация осуществляется путем корректировки буфера клавиатуры. Ниже, на Рис. 17.3, приведена готовая процедура имитации нажатия клавиши.

```
TO_BUF    PROC
          PUSH  ES
          PUSH  BX
          PUSH  DI
          MOV   BX,40H
          MOV   ES,BX
          MOV   BX,1CH
```

```
        MOV   DI,ES:[BX]
        MOV   ES:[DI],CL
        MOV   ES:[DI+1],CH
        CMP   WORD PTR ES:[BX],60
        JNZ   JJ
        MOV   WORD PTR ES:[BX],30
        JMP    SHORT  JJ1
JJ:
        ADD   WORD PTR ES:[BX],2
JJ1:
        POP   DI
        POP   BX
        POP   ES
        RET
TO_BUF    ENDP
```

Рис. 17.3. Процедура вставки в буфер клавиатуры символа. Код ASCII должен находиться в регистре CL, а скан-код в регистре CH.

В главе 7 приведена структура буфера клавиатуры. Напомню эту информацию. Ячейки - указатели на хвост буфера и на голову расположены соответственно по адресу 40H:1CH и 40H:1AH. Значение этих ячеек находится между 30 и 60. По достижению содержимого ячейки хвоста значения 60 следующее значение будет 30. В остальных случаях это значение увеличивается на 2. Буфер пуст, если значение хвоста и головы совпадает. Буфер переполнен, если содержимое ячейки-хвоста на 2 меньше содержимого ячейки-головы. Частным случаем переполнения является случай, когда хвост равен 60, а голова 30. На Рис. 17.3 показан упрощенный механизм вставки символа в буфер клавиатуры. Этот механизм игнорирует случай переполнения буфера. Механизм не приемлем для обработки нажатий клавиш, но вполне пригоден для обработки прерываний от мыши. Впрочем, для тех же самых целей Вы можете использовать одну из функций прерывания 16H (функция 4). Все дело сводится к тому, чтобы поместить в регистр CX правильное значение и вызвать процедуру TO_BUF.

При вызове процедуры прерывания содержимое регистров полностью определяет состояние мыши. В том числе и то, как двигалась мышь со времени последнего вызова процедуры прерывания (см. функцию 12).

В заключение данного раздела хочу акцентировать Ваше внимание на том, что прежде чем использовать драйвер мыши, необходимо сначала проверить, присутствует он или нет. Данная процедура является дальней и проверяет наличие драйвера мыши, а также возможность работы с мышью по следующим трем признакам:

1) вектор 33H не нулевой, 2) вектор не указывает на IRET (код CFH) 3) начальная установка драйвера мыши прошла успешно (функция 0). Если с драйвером мыши работать можно, то процедура возвращает 1, в противном случае 0. При выполнении этой процедуры происходит начальная установка драйвера (см. функцию 0).

```
PROV_MOUS  PROC  FAR
       XOR   AL,AL
       PUSH  ES
       PUSH  BX
       XOR   BX,BX
       MOV   ES,BX
       MOV   BX,ES:[33H*4]
       MOV   ES,ES:[33H*4+2]
       CMP   BX,0
       JNZ   CONT
       PUSH  BX
       MOV   BX,ES
       CMP   BX,0
       POP   BX
       JZ    NO
CONT:
       CMP   BYTE  PTR ES:[BX],0CFH
       JZ    NO
       XOR   AH,AH
       INT   33H
       CMP   AX,-1
       JNZ   NO
       MOV   AL,1
NO:
       POP   BX
       POP   ES
       RETF
PROV_MOUS  ENDP
```

Рис. 17.4. Процедура проверки присутствия драйвера мыши. Возвращает 1, если можно работать с драйвером.

IV. Вариант без курсора.

Наиболее простая возможность использования мыши в своих программах - это бескурсорный вариант. Удобство этого варианта заключается в том, что, если Ваша программа первоначально не предусматривала использование мыши, реализовать его не составит никакого труда. В начале программы лишь требуется провести инициализацию: начальную установку, установку процедуры прерывания, сделать курсор невидимым. В конце программы необходимо снять обработчик прерывания.

```
INT_MOUS   PROC
PUSHF
PUSH  AX
PUSH  BX
```

```
PUSH  CX
PUSH  DS
MOV   AX,DATA
MOV DS,AX ;здесь обработка двух нажатых кнопок
      TEST  BL,1
      JZ    N1
      TEST  BL,2
      JZ    N1
      XOR   DS:PRIZ,255 ;признак направления
      CALL  SOUND
;конец обработки двух нажатых кнопок
      JMP   KON
N1:
      TEST  BL,1
      JZ    N2
;имитация клавиши ENTER
      MOV   CL,13
      MOV   CH,28
      CALL  TO_BUF
      JMP   KON
N2:
      TEST  BL,2
      JZ    N3
;имитация клавиши ESC
      MOV   CL,27
      MOV   CH, 1
      CALL  TO_BUF
      JMP   KON
N3:
      CMP   DS:PRIZ,255
      JZ    N4
;горизонтальное движение
      OR    SI,SI
      JZ    KON
      JNS   N5
      MOV   CL,0
      MOV   CH,75
      CALL  TO_BUF
      JMP   SHORT KON
N5:
      MOV   CL,0
      MOV   CH,77
      CALL  TO_BUF
      JMP   SHORT KON
```

```
;вертикальное  движение
N4:
        OR    DI,DI
        JZ    KON
        JNS   N6
        MOV   CL,0
        MOV   CH,80
        CALL  TO_BUF
        JMP   SHORT KON
N6:
        MOV   CL,0
        MOV   CH,72
        CALL  TO_BUF
;сбросить счетчик сигналов микки
KON:
        MOV   AX,0BH
        INT   33H
        POP   DS
        POP   CX
        POP   BX
        POP   AX
        POPF
        RETF
INT_MOUS ENDP
```

Рис. 17.5.Процедура обработки прерывания от мыши.

Суть такой обработки заключается в имитации мышью следующих клавиш: клавиш управления курсором (влево, вправо, вверх, вниз), ENTER, ESC. Например, передвижение по меню, выбор пункта меню (ENTER) и откат (ESC). Причем в этом подходе мышь может работать в двух режимах - фиксация горизонтального движения и фиксация вертикального движения. Переход между этими двумя состояниями осуществляется одновременным нажатием обеих кнопок мыши (нажатие одной кнопки при нажатой другой). Режимы определяются содержимым переменной PRIZ. При переключении режимов раздается звуковой сигнал - вызов процедуры SOUND (см. главу 6, Рис. 6.5).

V. Курсорный вариант.

Наиболее простой вариант с курсором заключается в том, что "отлавливается" только нажатие на левую и правую кнопки. Правая кнопка, как и обычно, может имитировать нажатие клавиши ESC. Левая кнопка может имитировать клавишу ENTER, но это не всегда удобно. Иногда нужно отличать нажатие кнопки мыши от нажатия на клавишу клавиатуры. Поэтому я бы рекомендовал по нажатию левой кнопки отправлять в буфер клавиатуры какой-нибудь редко употребляемый код, например 1, 2 и т.п. На Рис. 17.6 представлена примерная процедура прерывания.

```
INT_MOUS   PROC
        PUSHF
        PUSH  DS
        PUSH  AX
        PUSH  BX
        PUSH  CX
;код обработки
        MOV   AX,0003H
        INT   33H
        MOV   AX,SEG X_MOUS
        MOV   DS,AX
        MOV   X_MOUS,CX
        MOV   Y_MOUS,DX
        TEST  BX,1
        JZ    NO_ENT
        MOV   CL,11
        MOV   CH,27
        CALL  IN_BUF
        JMP   SHORT  NO_ESC
NO_ENT:
        TEST  BX,2
        JZ    NO_ESC
        MOV   CL,27
        MOV   CH,1
        CALL  IN_BUF
NO_ESC:
;конец кода обработки
        POP   CX
        POP   BX
        POP   AX
        POP   DS
        POPF
        RETF
INT_MOUS   ENDP
```

Рис. 17.6. Процедура обработки прерывания от мыши. Курсорный вариант.

Относительно данной процедуры следует сделать следующие замечания:

1. X_MOUS и Y_MOUS являются глобальными переменными, которые после нажатия кнопки мыши содержат координаты курсора. Разумеется, в начале программы они должны быть проинициализированы. Если Вы работаете с текстовым курсором, то придется масштабировать значения координат, т.к. драйвер возвращает их пиксельное значение. Для текстового режима Вам следует поделить эти значения на 8. Регистр DS направляется на сегмент, где хранятся переменные X_MOUS и Y_MOUS.

2. В данной процедуре по нажатию правой кнопки в буфер клавиатуры возвращается код 11. В принципе Ваша процедура может возвращать разные коды в зависимости от некоторого дополнительного условия. То же можно сказать и относительно правой кнопки.

3. Разумеется, Ваша процедура прерывания может производить и непосредственные действия. Например, нажатие левой кнопки в каком-либо определенном месте экрана может вызывать какое-то действие: гашение экрана, смену типа курсора и т.п.

Здесь приведен довольно простой вариант взаимодействия с мышью. В некоторых случаях необходимо, чтобы программа мгновенно реагировала на положение курсора мыши. Соответственно Вам придется написать процедуру прерывания, которая вызывалась бы не только по нажатию кнопок, но и при передвижении мыши.

VI. Некоторые советы.

1. Одна из проблем, возникающих при программировании мыши, - это несоответствие скорости мыши (движение курсора, реакция на нажатие кнопки) потребностям программы. В этой связи обращаю Ваше внимание на функции 15, 19, 26, 28. Кроме того, можно сделать задержку в самой процедуре прерывания либо в самой программе после обработки нажатия кнопки мыши с последующей очисткой буфера клавиатуры.
2. В программе часто приходится обновлять содержимое экрана. Вам придется перед этим действием погасить курсор, а потом сделать его видимым.
3. При работе с функцией 3 надо иметь в виду, что результат выдается относительно прямоугольника 640*200. В зависимости от режима экрана вы должны привести эти координаты в соответствие с реальным разрешением экрана. Например, в стандартном текстовом режиме 80*25 значения координат следует умножить на 8: 80*8=640, 25*8=200. Не забывайте также, что курсор имеет размеры, а координата возвращается для его левого верхнего угла.
4. Некоторые программы используют так называемый двойной щелчок клавиши мыши. Суть его заключается в том, что какая-либо функция программы вызывается только тогда, когда в течение определенного промежутка времени дважды нажата какая-либо клавиша, при этом курсор мыши должен находиться в данной области. Я надеюсь, что читатель, дойдя до данной главы, самостоятельно сможет реализовать алгоритм, который я изложу ниже:
   а) поскольку второй щелчок должен произойти в некоторый промежуток времени, то Вам не миновать использования перехвата какого-либо временного вектора 09Н или 1СН
   б) при первом щелчке устанавливается флаг
   в) процедура прерывания по времени начинает отчет промежутка
   г) если промежуток прошел, а второго щелчка не было, то флаг сбрасывается
   д) второй щелчок идентифицируется по двум параметрам - установлен флаг, и курсор мыши находится в данной области экрана
   е) если флаг установлен, но курсор уже находится в другой области, то флаг должен быть сброшен
   ж) наконец, если областей несколько, то вместе с флагом придется устанавливать идентификатор области.

Глава 18. Элементы теории вирусов.

- *Нет, я его вычесывал.*
- *А отчего же блохи?*
- *Не могу знать. Статься может, как-нибудь из брички поналезли.*

Н.В. Гоголь.
Мертвые души.

Уважаемый читатель, автор просит прощение за несколько общий и конспективный характер изложения материала. Дело в том, что компьютерная вирусология уже давно выделилась в отдельную отрасль со своими законами и технологиями. Объем же главы не позволяет углубиться в этот интересный и волнующий материал более детально. Возможно, в последующих изданиях книги я расширю объем излагаемого материала.

I. Предварительные замечания.

Социальные причины появления компьютерных вирусов - вопрос довольно сложный. Для кого-то это способ самоутверждения, а кто-то просто учится на них программировать (это интереснее, чем писать архиватор или текстовый редактор). Наконец, в появлении вирусов заинтересованы программисты и фирмы, производящие антивирусные программы. Наверное, можно назвать еще несколько причин, но этим должен заниматься психолог, а не программист. Мне, как и любому автору, пишущему о вирусах, хотелось бы высказать мое видение этой проблемы.

Я не согласен с теми, кто считает, что в деле борьбы с вирусами все уж так безнадежно. Компьютерные вирусы есть, и число их множится, но одновременно растет число антивирусных средств. Конечно, создатели вирусов идут несколько впереди, но не такой уж это большой разрыв. Так ли все это плохо? Представьте себе на секунду, что с земли неожиданно исчезли все болезнетворные микробы. Что в этом случае произошло бы с иммунной системой человека? Она атрофировалась бы. После этого человечество могло бы вымереть просто от насморка - ведь нет же гарантий, что микробы, исчезнув, не появятся вновь.

По-видимому, на развитии вирусов сказались следующие факторы:

1. Резкое увеличение парка ЭВМ в основном за счет персональных компьютеров. Это создало благоприятную среду для распространения компьютерных вирусов.
2. Большое значение для распространения вирусов имеет все большее использование локальных и глобальных компьютерных сетей. Компьютерная сеть представляет собой уже некое новое образование со своими свойствами и, увы, слабыми средствами защиты.
3. Выросло и продолжает расти количество людей, работающих в сфере информационных технологий, - среда, из которой выходят будущие создатели вирусов.
4. Все более усложняющаяся программная среда, стоящая между человеком и компьютером. Вероятно, на каком-то этапе развития таких сред появление (даже

11 – 4072

случайное) саморазмножающихся алгоритмов неизбежно. Рост уязвимости компьютерных систем наводит на мысль, что надежность в будущем станет одним из главенствующих требований к программному обеспечению.

Что же тогда делать? Да то, что делают во всем мире: совершенствовать законы, разрабатывать новые антивирусные средства, организовывать специальные службы по компьютерным преступлениям, создавать новые операционные системы. А появление вирусов - вещь неизбежная. Компьютерная преступность, как и любая другая, имеет и социальные корни, и, на мой взгляд, эти корни в значительной степени идентичны.

Проблема компьютерных вирусов имеет и правовой аспект. В какой, например, степени можно считать виновным человека, написавшего программу-вирус? Или преступлением следует считать лишь способствование его распространению? Все это интересные вопросы, но автор не считает себя компетентным настолько, чтобы заниматься дальнейшим их обсуждением.

Не могу не остановиться еще на одном аспекте, связанном с компьютерными вирусами. Нет слов, написание вируса и его распространение - акт не совсем нравственный. Но как следует расценивать тот факт, что антивирусные программы сопровождаются утверждениями, что автором того или иного вируса является тот или иной человек. Это, на мой взгляд, является еще более безнравственным, т.к. обвинительное заключение выносит, как известно суд. На каком, спрашивается, основании делаются столь безответственные утверждения. Оказывается, например, на основании того, что в теле вируса имеется строка с зашифрованной фамилией (или координатами) автора. Авторы антивирусов грешат отсутствием элементарной логики. Разве фамилию в теле вируса (и любой другой программы) может поставить только автор? Для того чтобы утверждать авторство данного вируса, нужны чисто юридические доказательства.

Конечно, операционная система MS DOS весьма предрасположена к созданию вирусов. Этому способствуют отсутствие средств разделения доступа к ресурсам, большое количество недокументированных способов (тем не менее известных) обхода стандартных процедур. Однако я категорически не согласен с теми, кто приводит в пример другие операционные системы, аргументируя малым количеством вирусов, существующих для них. Сравните количество компьютеров, работающих с операционной системой MS DOS или Windows, с количеством компьютеров, работающих под управлением других операционных систем, и все станет ясно. Здесь работает закон больших чисел. Кому же взбредет в голову писать вирус, работающий в среде UNIX, если в округе этой системой никто не пользуется. Для авторов вирусов, несомненно, важен тот факт, что его детище живет и распространяется. С распространением операционной системы Windows стали множиться и вирусы, работающие в этой системе. Число их в настоящее время уже достаточно велико. В настоящее время оно уже сравнимо (с учетом вирусов, поражающих документы Word и Exel) с вирусами для MS DOS.

Мысли из предыдущего абзаца не претендуют на строгость и высказаны в плане дискуссии. Главный вопрос, по-видимому, здесь в том, могут ли вирусы затормозить или существенно повлиять на развитие информационных технологий. Возможно, в ближайшее время мы будем свидетелями наступления некоторого динамического равновесия. Главную роль здесь будут играть, по-видимому, профилактические средства, а не средства лечения. При создании как системного, так и прикладного программного

обеспечения значительное внимание будет уделяться защите программ и систем в целом. Изменится психология как коллективных, так и индивидуальных пользователей. Значительную роль сыграют административные меры защиты.

Некоторые определения.

Для того чтобы рассматривать свойства компьютерных вирусов, дадим предварительно несколько простых определений.

Под компьютерным вирусом будем понимать программный модуль (группу модулей), способный модифицировать другие программные модули так, чтобы получать управление во время работы последних, и обладающий способностью размножаться, т.е. производить программные модули, способные производить аналогичные действия.

Введем понятие активного состояния вируса.

Выполняющийся вирус будем считать активным.

Вирус, находящийся в оперативной памяти, будем считать активным, если он способен получить управление при выполнении каких-либо программ.

Программу будем считать зараженной, если ее запуск (выполнение) может привести к активизации вируса.

Гибкий, жесткий диск (или другой носитель) будем считать зараженными, если на них имеются зараженные программы.

Компьютер с активным вирусом либо зараженным жестким диском будем считать зараженным.

Сигнатура – это строка в теле вируса, по которой можно осуществлять его поиск на диске или в оперативной памяти.

Распространение глобальных компьютерных сетей не требует особой корректировки в вышеизложенных положениях. Однако в компьютерных сетях возможны вирусы, которые распространяются по сети, не используя при этом жестких дисков. Во всяком случае, представьте ситуацию, когда вирус передается по сети, не успев заразить еще ни один компьютер. Такие вирусы иногда называют червями. Очевидно, что для них следует вводить понятие "зараженная сеть" или сегмент сети, но такие ситуации мы в данной главе не рассматриваем.

II. Файловые вирусы в MS DOS.

По одной из классификаций вирусы делят на файловые и загрузочные (бутовые). К файловым вирусам относят вирусы, поражающие запускаемые файлы (программы), а к загрузочным — вирусы, активизирующиеся через главную загрузочную запись или через BOOT-сектор. Деление довольно условное, так как стали появляться вирусы, сочетающие в себе свойства тех и других.

Мы, однако, традиционно разделим вирусы, обитающие в среде MS DOS на файловые и бутовые (загрузочные). В разделе IV мы поговорим о других видах.

Файловые вирусы (ФВ) могут заражать как COM так EXE-программы. Ниже перечислены стандартные способы заражения [45].

---

[45] Часто авторы вирусов прибегают к весьма изощренным методам заражения.

1. Способы заражения СОМ-программ. На Рис. 18.1 представлены три способа заражения СОМ-программ. Легко заметить, что заражение осуществляется так, чтобы вирус получал управление сразу после запуска программы. Как Вы понимаете, команда JMP символизирует здесь просто передачу управления. Вместо нее с тем же успехом пользуются командами RET или CALL с соответствующей корректировкой стека.

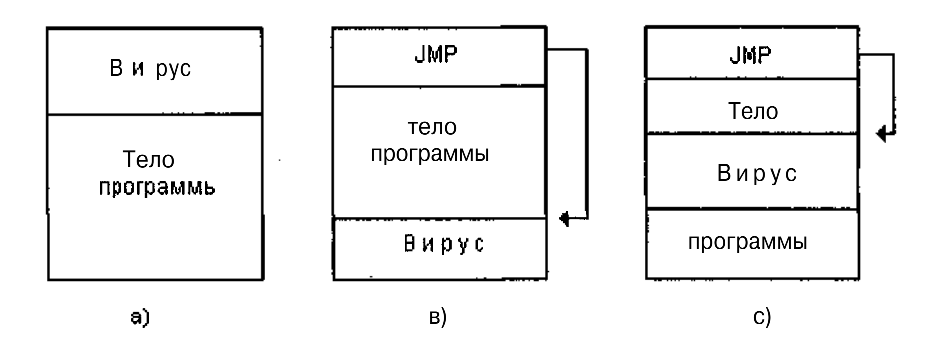


Рис. 18.1. Способы заражения СОМ-программы.

В конце главы будет приведен пример "заражения" программы по типу b.

2. Заражение ЕХЕ-программ. Заразить ЕХЕ-программу несколько сложнее, хотя идея заражения довольно ясна. Тело вируса приписывается к какой-либо части программы, а в заголовке меняются значения регистров CS и IP (см. Главу 14). Таким образом, при запуске программы управление вначале опять передается вирусу. Наиболее изощренные вирусы корректируют еще значения SS и SP, дабы не угодить в область стека, а также проверяют программу на наличие в ней оверлеев. Наличие в программе оверлея может привести к тому что управление никогда не будет передано вирусу, и программа перестанет запускаться.
3. Некоторые вирусы способны заражать оверлеи (см. главу 11), имеющие стандартную структуру. Активизация вируса происходит при запуске оверлея прикладной программой.

   Появившийся некоторое время назад вирус DIR является своего рода качественным развитием файловых вирусов. Данный вирус корректирует FAT так, что при запуске любой программы управление передается ему. После того как он оказался в памяти, стандартные утилиты уже не могут обнаружить искаженную структуру FAT.
4. Активизация вирусов. Часть вирусов, которые называют не резидентными, активизируются лишь временно, во время запуска зараженной программы. Как правило, за время активизации они пытаются заразить одну или несколько программ. Поиск программ для заражения может производиться в текущем каталоге, корневом каталоге, через PATH в окружении или посредством поиска по всему дереву каталогов (алгоритм достаточно сложен для реализации его на ассемблере, и, кроме того, для осуществления поиска требуется время).

Вирусы, называемые резидентными, активизируются во время запуска программы и остаются активными в течение всего времени работы компьютера. Некоторые вирусы остаются активными даже после перезагрузки через Ctrl Alt Del, т.к. обрабатывают соответствующее прерывание.

Рассмотрим подробнее, как вирусы устанавливают себя в памяти. При запуске программы управление передается вирусу. В его задачу входит: а) установить себя в память; б) передать управление запущенной программе. Есть три области памяти, где может прижиться вирус. Это старшие адреса памяти, младшие адреса памяти и системная область. Установка вируса посередине адресного пространства неэффективна, т.к. сразу же даст о себе знать - перестанут запускаться программы. Остаться резидентным в старших адресах памяти наиболее простая задача. Вирус перемещает себя в нужную область памяти, а затем корректирует текущий МСВ, уменьшая размер блока на размер захваченной области. Сложнее остаться резидентным в младших адресах памяти. Свободного блока может не быть, и придется перемещать программу. Если это ЕХЕ-программа, то не избежать корректировки адресов. Кроме того, придется правильно создать свой МСВ и, следовательно, скорректировать предыдущий. Зато вирус в младших адресах памяти не так бросается в глаза. Некоторые вирусы "прикрепляют" себя к телу легальной резидентной программы, что значительно затрудняет их обнаружение. Если вирус не слишком большой, то сможет найти себе место и в системной области. Это может быть системный стек, системный буфер (тогда должно быть уменьшено число системных буферов), область данных DOS и BIOS или область векторов прерываний. Отметим, что есть вирусы, которые, как ни странно, для установки себя в памяти используют стандартное прерывание 27Н. Оказавшись в памяти, такой вирус еще раз запускает зараженную программу, оставляя занятым лишь блок, который он занимает. Передает управление сразу на нее, а по выходу из этой программы передает управление DOS через прерывание 27Н (!). Заметим также, что вирус может состоять из нескольких частей, которые могут располагаться в разных областях памяти и взаимодействовать друг с другом.

Вирус, помещая себя в память, перехватывает некоторые прерывания, чтобы контролировать ситуацию и иметь возможность размножаться. Как правило, это 21-е и 13-е прерывания. Посредством этих прерываний можно контролировать обращение системы к диску и при благоприятной возможности заразить программу. Чаще всего для этой цели используется функция DOS 4ВН, вызываемая каждый раз для запуска программы[46]. Есть, однако, вирусы, которые заражают файл даже при его открытии или чтении. Конечно, легко написать программу, которая путем отслеживания прерываний 13Н и 21Н фиксировала бы подозрительные действия, происходящие в системе (такие программы называют мониторами). Однако многие вирусы трассируют прерывания, определяя значения векторов до того, как их перехватили какие-то программы, или же используют недокументированные точки входа в системные процедуры. Поэтому такие программы, даже запущенные до активизации вируса, не способны отслеживать его действия. Возможно также, что вирус направит вектор в область памяти,

[46] Некоторые антивирусные программы проверяют подозрительные области памяти на наличие команды CMP AH,4BH.

где будет находиться небольшая процедура, которая нестандартным способом (например, через IRET) передаст управление вирусу, который будет находиться в другой области пространства.

Перехваченные векторы могут приводить, разумеется, к проявлению как разрушительных (форматирование дискет, порча файлов, зависание и т.п.), так и развлекательных (картинки, шуточные сообщения, звуковые проявления и т.д.) функций вируса.

К файловым вирусам следует отнести и вирусы-спутники. Действие этих вирусов весьма остроумно. Суть в том, что рядом с EXE-программой создается COM-программа, но с таким же именем. Понятно, что это есть в чистом виде сам вирус. Когда же мы запускаем EXE-программу, то по правилам операционная система ищет вначале COM-модуль и запускает его. Активизируясь, вирус затем запускает и саму программу. После этого запуск незараженной программы вызывает появление в том же подкаталоге ее COM-двойника. В более простом варианте вирус является нерезидентным, и заражение происходит во время запуска программы.

III. Бутовые (загрузочные) вирусы.

На столе лежит дискета
У нее испорчен boot.
Через дырочку в конверте
Ее вирусы грызут.

Программистский фольклор.

Стандартный механизм заражения такими вирусами таков: вместо программы загрузки подставляется другая программа, которая:

1. При запуске системы вначале загрузит в память резидентную часть вируса.
2. Перенаправит нужные векторы прерываний на эту резидентную часть.
3. Запустит программу загрузки так, чтобы процесс загрузки продолжался.

Все такие вирусы, как правило, работают с прерыванием 13H, которое перехватывают при загрузке системы. Хитрость здесь в том, чтобы обманывать программы, просматривающие диск. При попытке просмотреть BOOT-сектор вместо него подставляется истинный, без вируса.

Часто одного сектора недостаточно для тела вируса, и другую свою часть он прячет в свободных секторах, которые можно найти в начале, либо в секторах, которые затем он помечает как сбойные. Свободные секторы найти совсем несложно. Действительно, вся дорожка цилиндра 0 стороны 0 пустует, кроме первого сектора, где располагается Partition Table или BOOT-сектор (для дискеты). На дискетах бутовые вирусы прячутся также в последних секторах корневого каталога. Наконец, вирусы могут содержать в себе свой загрузчик. Такие вирусы при записи в BOOT-сектор затирают старый загрузчик. Некоторые вирусы (знаменитый вирус BRAIN) используют свободные кластеры на диске, которые ищут по FAT-таблице, помечая их как испорченные.

Бутовых вирусов не так много, как файловых, и распространяются они не так быстро. Связано это с тем, что для заражения компьютера требуется попытка запуска с зараженной дискеты. Однако в последнее время начинают появляться вирусы, сочетающие в себе свойства как файловых, так и бутовых вирусов.

Ниже представлен дезассемблированный фрагмент такого вируса, называемого 6-м марта (Рис. 18.2). Текст всего вируса не приводится из этических соображений. Вирус находится в MBS винчестера или в BOOT-секторе дискеты. Истинные же секторы хранит в другом месте. BIOS считывает этот сектор по адресу 0000:7C00H и передает туда управление.

```
        ORG  0
        JMP  BEG_VIRVIR_OFF
        DW   BEG_1FREE
        DW   ?     сегментный адрес, куда будет скопирован
                  ;вирус, одновременно это размер свободной
                  ;памяти в параграфах с резервированием
                  ;места для вируса
        .
        .
13_OFF DW   ?          ;адрес обработчика прерывания 13H
13_SEG DW   ?
INT_13 PROC            ;обработчик прерывания 13H
        .
        .

BEG_VIR:
        XOR AX,AX
        MOV DS,AX
;установка стека
        CLI
        MOV SS,AX
        MOV AX,7C00H
        MOV SP,AX
        STI
;4-байтный адрес в стек для последующего
; скрытого (RETF) перехода на истинную программу
;загрузки
        PUSH DS
        PUSH AX
;запомнить адрес обработчика прерывания 13H
        MOV  AX, [13H*4]
        MOV  13_OFF,AX
        MOV  AX, [13H*4+2]
        MOV  13_SEG,AX
;уменьшить объем свободной памяти на 2 килобайта
        MOV  AX,[413H]
        DEC  AX
        DEC  AX
        MOV  [413H],AX
```

```
;перевести объем в параграфы
      MOV   CL,06
      SHL   AX,CL
      MOV   ES,AX
;запомнить сегментный адрес
      MOV   FREE,AX
;установить собственный обработчик прерывания 13H
      MOV   AX,OFFSET INT_13
      MOV    [13H*4],AX
      MOV    [13H*4+2],ES
;подготовить копирование тела вируса в конец памяти
;чтобы по адресу 0000:7C00H считать истинную программу
;загрузки
      MOV   CX,1BEH        ;размер вируса
      MOV   SI,7C00H       ;откуда
      XOR   DI,DI
      CLD
      REPZ MOVSB
;переход в тело вируса, которое уже скопировано в новое место
      JMP   DWORD PTR CS:VIR_OFF
BEG1:
      .
      .
```

Рис. 18.2. Фрагмент вируса 6 марта.

На Рис. 18.2 показано начало работы вируса. Далее идут также достаточно прозрачные вещи: из укромного места считывается программа загрузки, и ей передается управление. В дальнейшем вся деятельность вируса будет связана с работой 13Н-го прерывания. В частности, вирус должен будет обеспечить свое размножение - запись на гибкие и жесткие диски.

IV. Другие вирусы.

Существуют вирусы, встречающиеся гораздо реже, чем файловые или бутовые вирусы. К таким, в частности, относятся вирусы, которые можно назвать драйверными. Они работают на уровне загружаемых драйверов устройств, внедряясь в системную область после загрузки операционной системы либо используя для заражения файл CONFIG.SYS. Последнее довольно интересно, т.к. использование строки 'DEVICE=' требует, чтобы загружаемый драйвер находился на диске в виде файла. Здесь возможны различные уловки, например, вирус вначале активизируется через программу загрузчик, подставляет необходимую строку в CONFIG.SYS и образует соответствующий драйвер. Далее загрузка идет своим чередом, и вирус оказывается загруженным в область драйверов. После загрузки, естественно, строка в CONFIG.SYS и драйвер в конце уничтожаются. Другие вирусы внедряются непосредственно в драйвер устройства. Стандар-

тный механизм заражения драйвера таков: вирус приписывает себя к концу драйвера и модифицирует адреса программы стратегии и программы прерываний (см. главу 16) так, чтобы при вызове драйвера обращение вначале происходило к вирусу.

Определенную группу вирусов называют "червями". Эти вирусы орудуют в компьютерных сетях. Переходя от компьютера к компьютеру, они могут практически не использовать дисковое пространство и работать только на уровне оперативной памяти. Существование таких вирусов для локальных IBM-сетей пока неизвестно.

До сих пор мы говорили о вирусах, "работающих" на уровне машинных команд. К таковым относятся и вирусы, написанные на языках высокого уровня (есть такие вирусы, работающие в среде MS DOS). В любом случае сам механизм встраивания в прикладную или системную программу происходит на уровне команд микропроцессора. Однако возможен совершенно иной тип вируса (автору неизвестно о существовании таких вирусов на персональных компьютерах). Такие вирусы можно назвать командными. Принципиально нет ничего, что могло бы помешать их появлению[47].

Современные прикладные пакеты (редакторы, электронные таблицы, системы управления баз данных и т.п.) часто имеют свои языки управления заданиями. Эти языки могут быть весьма мощными и гибкими, в принципе позволяющими создавать саморазмножающиеся объекты. Похожая ситуация складывается и в среде операционных систем. В MS DOS существует возможность написания пакетных файлов - выполняемых программ, состоящих из команд операционной системы. Существенно то, что вместо команд операционной системы может стоять любая программа - обычный загрузочный модуль, выполняющийся в среде операционной системы. Таким образом мы можем до бесконечности пополнять и совершенствовать набор стандартных команд. Не составит труда написать программу, которая бы разыскивала BATCH-файлы и дописывала бы к ним несколько строк.

Наконец, принципиально возможен еще один тип вирусов. Такие вирусы можно назвать транслируемыми. К тексту программы на языке высокого уровня дописывается некоторое количество строк. После трансляции вирус оказывается неотъемлемой частью программы. Процесс же дописывания строк к текстам программ будет продолжаться.

V. Вирусы в операционной системе Windows.

С появлением операционной системы Windows 95 поле деятельности производителей компьютерных вирусов в значительной степени расширилось. Здесь бы я выделил три типа вирусов, которые так или иначе связаны с новой операционной системой:

1. Вирусы, написанные на Visual-Basic и внедряющиеся в файлы, создаются для MSWORD и MSEXEL. Эпидемия этих вирусов прокатилась некоторое время назад, в настоящее же время они стали неотъемлемой частью нашей жизни. Принцип функционирования этих вирусов весьма прост. Многие программы создают файлы, которые могут содержать в себе макрокоманды. Причем можно сделать так, что эти макрокоманды будут выполняться каждый раз перед загрузкой этого файла (макрос AutoOpen). Макро-вирус вставляет свою процедуру так, что она выполняется перед загрузкой документа. Поэтому образом вирус будет каждый раз получать управление. Все остальное — дело техники.

[47] Операционная система UNIX имеет весьма развитый командный язык.

2. Вирусы, в целом написанные для MS DOS, но использующие так или иначе наличие операционной системы Windows. Здесь речь может идти о самых разных проявлениях. Есть вирусы, поражающие EXE-файлы в формате NE. Причем DOS-часть этих файлов. Так что они активизируются только при запуске этих файлов в MS DOS. Есть вирусы, встраивающиеся в загрузочные файлы Windows и т.д.
3. Полноценные Windows-вирусы. Пока преобладает, скажем так, не резидентный подход. Заражается EXE-файл (формат NE или PE). При запуске такой программы вирус активизируется на время, необходимое для поиска программы-жертвы. Появились также вирусы с иным механизмом поражения. В частности, запись вируса в виде VXD-файла. Вирус также корректирует SYSTEM.INI, так чтобы данный "драйвер" загружался каждый раз при загрузке Windows. SYSTEM.INI используется также для заражения программ, всегда загружаемых при запуске WINDOWS. Вообще понятие резидентной программы в Windows потеряло свой смысл. Любая программа может быть резидентной, т.е. находится в памяти при запуске и работе других программ. Следовательно, любой вирус, заразивший ту или иную программу и не удаляющий себя из памяти при запуске, может быть резидентным. Вопрос заключается лишь в том, может он активизироваться или нет при активизации зараженной программы.

VI. Борьба с вирусами.

- Скажите мне, - спросил Руневский, - каким образом вы узнаете, кто упырь, а кто нет?
- Это совсем немудрено.

А.К. Толстой.
Упырь.

- Меры вот какие. Взял я на кухне свечечку...

М.А. Булгаков.
Мастер и Маргарита.

Мы попробовали провести некоторую классификацию проявлений вирусов. По этим признакам можно судить о присутствии вирусов на диске или в памяти, не имея специальных средств:

а) Изменение длины программ. Не многим вирусам удается не менять длины программы (см. ниже). Некоторые вирусы пытаются так подкорректировать длину, чтобы ее изменение было незаметно. Например, не садятся на слишком короткие программы, корректируют длину так, чтобы она изменилась на 1000 байт (не так заметно) и т.д.

б) Нехватка памяти. Некоторые программы сообщают о нехватке памяти, некоторые выдают сообщения об ошибке или зависают. В любом случае это должно Вас настorожить.

в) Когда вирус "садится" на программу, то происходит автоматическая корректировка времени и даты файла. Если вирус не восстанавливает старые значения, то это может служить хорошим признаком несанкционированного доступа к Вашей программе. Интересно, что установка во времени корректировки файла 62 секунды (проверьте, что установить 63 или 61 секунду невозможно, и объясните почему) является для многих вирусов признаком зараженности программы.

г) Проявления, связанные с развлекательными функциями вируса. Это могут быть необычные или шутливые сообщения, странные явления на экране (осыпание букв и т.д.), звуковые сигналы и музыка и т.д.

д) Проявления, связанные с разрушительными функциями вирусов. Это может быть неожиданно переставшая запускаться программа, испорченная база данных, не запускающаяся дискета, странные зависания системы и т.д.

е) Проявления, связанные с некорректной работой вируса или наличием ошибок в нем. Классическим примером может служить отсутствие обработки критических ошибок для MS DOS (см. Глава 9). Некоторые вирусы, заражая программу, безнадежно портят ее (иногда ошибка, а иногда и намеренные действия). Часто вирусы некорректно работают с операционной системой. Например, не работают на DOS 6.О. Портят содержимое регистров при входе в программу. Проявлений здесь огромное количество. Некорректная загрузка Windows также должна Вас насторожить.

ж) Иногда о наличии в программе вируса можно судить по присутствию в нем некоторых характерных строк, которые можно обнаружить даже при "текстовом" просмотре. Вот эти типичные строки: *.COM, *.EXE, PATH=, COMMAND. "Умные" вирусы, однако, шифруют строки, подобные этим.

з) Нерезидентные вирусы после запуска зараженной программы должны найти новую жертву, прежде чем передать управление программе. Поиск, вообще говоря, может потребовать времени. Поэтому отмечайте для себя те случаи, когда некоторые программы начнут запускаться медленнее обычного.

Появление новых операционных систем, таких, как Windows, вносит серьезные коррективы в дело обнаружения вирусов. Сразу скажу, что обнаружение вирусов способами, описанными выше, становится все менее и менее возможным. Из-за огромного количества файлов как в самой операционной системе, так и в используемых пакетах, никто уже не просматривает списки файлов. Да и сами программы запускаются или с рабочего стола, или из меню. Наличие же ошибок в операционных системах Windows 95 и Windows 98 приучает пользователей к не совсем корректной работе системы. Зависания, некорректные загрузки и неожиданные перезагрузки компьютера теперь списывают на систему, а о вирусах забывают. Все это достаточно печально и наводит на мысль о том, что компьютерным вирусам далее будет жить все вольготнее. Наше время ознаменовано также широким распространением локальных и глобальных компьютерных сетей, что еще в большей степени усложняет возможность обнаружения вируса.

Сейчас наработан довольно большой арсенал программных средств защиты от вирусов. Ниже перечисляются эти средства защиты.

Антивирусные ревизоры. Суть работы этих программ заключается в том, что состояние программ, системных файлов, boot-секторов и т.п. запоминается. При запуске ревизора происходит сравнение контрольных сумм. При обнаружении изменений выдается сообщение или производятся, какие-либо другие действия. Разумеется, ревизор не может точно сказать, обнаружен вирус или изменения вызваны другими причинами. Основным требованием, предъявляемым к таким программам, является умение обнаружить изменения, производимые вирусом, даже если вирус находится в памяти. Ревизия должна производиться по крайней мере каждый раз после запуска операционной системы. Существуют резидентные ревизоры, которые могут проверять контрольную сумму программы перед самым ее запуском.

Детекторы. Эти программы рассчитаны на вполне определенные вирусы, которые ищутся по сигнатуре - строке, содержащейся в теле вируса. Некоторые детекторы позволяют пользователю пополнять список сигнатур. Использование таких программ достаточно ограниченно. Новые вирусы появляются с огромной быстротой, и даже авторы самых знаменитых и эффективных детекторов не поспевают за ними.

Фаги. Программы, позволяющие восстанавливать (лечить) зараженные файлы. Обычно такие программы имеют и встроенный детектор. Данные программы весьма опасны, т.к. часто портят восстанавливаемые файлы. Весьма незначительные изменения в вирусе могут привести к изменению длины вируса или отдельных его частей и даже способа заражения. После чего попытка излечить программу от данного вируса скорее всего приведет к порче программы.

Вакцины и вакцинация. Идея заключается в попытке обмануть вирус. Большинство вирусов перед тем как "сесть" на программу проверяет, не заражена ли уже она. Для этого в теле программы в определенном месте ищется специальная метка. Если в нужное место программы поместить эту метку, то программа тем самым будет защищена от данного вируса. Резидентная программа-вакцина находится в памяти и имитирует наличие там вируса. При запуске зараженной программы вирус проверяет наличие себя в памяти по определенным признакам. Вакцина обманывает вирус, не позволяя ему остаться в памяти. Легко сообразить, что количество вирусов, от которых можно уберечься таким способом, невелико. Трудно себе представить вакцинацию, скажем, от ста вирусов одновременно. Такой метод можно эффективно использовать лишь во время эпидемий на машинах со многими пользователями.

Резидентные сторожа. Сторож — это программа, позволяющая выявить или блокировать несанкционированные действия в системе. Таковым может быть либо заражение программы, либо попытка остаться в памяти резидентно. Отслеживанием прерываний и сравнением объема свободной памяти до и после запуска программы это сделать не так уж трудно. Проблема заключается в том, что запись на диск производится довольно часто, и сторож должен проявлять определенную "интеллектуальность", реагируя лишь на подозрительные операции.

Программа может иметь и своего собственного сторожа, который запускается при запуске программы и далее проверяет возможность ее заражения. В конце главы приводятся примеры того, как в принципе может быть построен такой сторож.

VII. О вирусных технологиях.

Происходит своеобразное соревнование между авторами вирусов и антивирусов. Остановлюсь лишь на двух пунктах этого соревнования: возможность контекстного поиска вируса на диске по какой-либо строке (сигнатуре), изменение видимой длины зараженного файла.

Начнем с возможности поиска вируса по сигнатуре. В первых статьях о вирусах сигнатуре уделялось большое внимание. Приводились таблицы наиболее удачных, по мнению авторов, сигнатур. Многие вирусы шифровали свое тело, но, поскольку, алгоритм шифрования не менялся, сигнатуры все равно существовали. Способ шифрования может быть достаточно прост - обычно используется команда XOR: две команды типа XOR AL,B , где B произвольный байт не меняют содержимое регистра AL. Легко видеть, что способ шифровки можно изменить, меняя значение байта B. Меняя случайным образом, способ шифровки, вирус каждый раз будет представать в новом обличье. Есть, однако, одно "но" - участок, который занимается расшифровкой, не может быть зашифрован, и, следовательно, по нему вирус может быть обнаружен и расшифрован. Казалось бы, вопрос с сигнатурой решен - она должна существовать для любого вируса. Однако существует подход, позволяющий создавать вирусы, поиск которых по сигнатуре невозможен. В простейшем варианте данный способ заключается в том, что значащие команды "разбавляются" незначащими, которые никак не влияют на работу вируса. Это могут команды NOP, MOV MEM,AX, PUSH AX/POP AX, CLD, STI и т.д. Причем количество незначащих команд может также варьироваться, и поэтому будет все время меняться длина вируса [48]. В более изощренном варианте команде или группе команд ставится в соответствие команда или группа команд, результат выполнения которых эквивалентен первой. Чтобы уяснить идею сказанного, ниже приводится таблица соответствия для некоторых команд или их групп команд[49].

В таблице приведены лишь три команды и три варианта к ним, но Вы можете продолжить эту таблицу как по вертикали, так и по горизонтали. Таким образом, можно получить несколько кодов программ, абсолютно не похожих друг на друга, но тем не менее эквивалентных. Наиболее действенный, на мой взгляд, результат может быть достигнут сочетанием шифрования тела вируса и модификацией его части, занимающейся расшифровкой. Вирусы, шифрующие свое тело, называют "вирусами-призраками".

Изменение длины зараженной программы является существенным признаком присутствия вируса. Однако авторы вирусов и здесь нашли возможность обойти эту проблему. Наиболее очевидным способом является перехват функций DOS FindFirst и

---

[48] Первые работы по компьютерной вирусологии придавали большое значение длине файловых вирусов.

[49] Вообще говоря, следовало бы дать определение эквивалентности команд или групп команд. Например, есть отличие между эквивалентностью команд MOV AX,BX и XCHG BX,AX/MOV BX,AX, с одной стороны, и команд MOV AX,BX и MOV MEM,BX/MOV AX,MEM - с другой. Во втором случае меняется содержимое ячейки MEM. Я думаю, дальнейший анализ читатель сможет провести сам.

FindNext и подстановка длины незараженного файла. Надо, однако, иметь в виду, что некоторые программы читают директорию напрямую, посредством INT 25H и их не обманешь. Более изощренный метод заключается в том, чтобы при заражении файла в директорию поставить длину незараженной программы, во время же запуска программы и при выполнении некоторых других функций DOS подставляется истинная длина. При этом, естественно, речь идет о ситуации, когда вирус находится в памяти. Казалось бы, все, но найден еще один поразительный способ. Идея, как ни странно, удивительно проста: файл можно заархивировать, и тогда его длина уменьшиться. К такому файлу приписывается вирус и необходимые байты, и длина зараженной программы не меняется. В более упрощенном варианте этот метод заключается в том, чтобы найти в файле цепочку повторяющихся байтов, которую можно заменить просто тройкой байт (все понятно, правда?), а на освободившееся место поместить вирус. Наконец, есть еще одна возможность маскировать свое присутствие в файле. Дело в том, что единицей записи на диск на уровне операционной системы является кластер. Длина кластера может составлять несколько секторов (см. главу 14). Если вирус не слишком длинен, он может помещать информацию (например, отрезок программы, который он занял) в свободные секторы последнего кластера, которого, вообще говоря, может и не быть. Длина программы при этом меняться не будет. Однако при копировании такие программы могут быть безнадежно испорчены.

MOV AX,BX	PUSH BX ЮРАХ	XCHGAX,BX MOV BX,AX	MOV MEM,BX MOV AX,MEM
JMPL1	MOV MEM,BX LEA BX,L1 PUSHBX MOV BX,MEM REIN	POP MEM CALL L1	LEA BX,L1 MOV MEM,BX LEA BX,MEM JMP[BX]
ADD AX,CX	PUSH CX L1: ADD AX, 1 LOOP L1 POP CX	MOV MEM,CX ADD AX,MEM	PUSHCX L1: ADD CX,1 SUB AX, 1 JNZ L1 MOV AX,CX POPCX

Вирус может быть обнаружен при визуальном просмотре диска или файла. Здесь обман пользователей происходит либо путем подстановки незараженного кластера, либо даже удалением вируса из файла перед просмотром. Последний вариант весьма неуклюж, однако, работает и создает определенные сложности при поиске вируса. Вирусы, маскирующие свое присутствие на компьютере, называют **"стелс-вирусами"**.

На этом я закончу рассуждения о вирусных технологиях. Замечу, что развитие их сделало бесполезным многие антивирусные программы или даже их виды. Например,

использование недокументированных точек входа в MS DOS делает бесполезной работу многих программ-сторожей. Заражение программ во время операций копирования делает весьма проблематичным обнаружение факта заражения.

VII.

И как пьяный сторож,
Выйдя за дорогу.

С. Есенин

Здесь приводится пример того, как к COM-программе можно подсоединить другую программу.[50] Эта программа может быть и вирусом, и сторожем, который при запуске будет проверять целостность программы (контрольную сумму и т.п.). Ниже приведены тексты трех программ:

Рис. 18.3 - программа подсоединяет к COM-программе программу-сторож (Рис. 18.5, имя PO.COM).

Рис. 18.4 - программа удаляет из COM-программы программу-сторож.

Рис. 18.5 - программа-сторож. При запуске COM-программы вначале получает управление сторож. Наша программа не делает ничего, лишь выводит строку и передает управление основной программе.

Приведенные программы демонстрируют некоторые простейшие технологии, о которых мы говорили выше.

```
;программа установки сторожа на COM-файл
CODE SEGMENT
      ASSUME CS:CODE, DS:CODE, SS:CODE
      ORG 100H
BEGIN:
;блок анализа командной строки
      XOR SI,SI
      XOR DI,DI
      MOV DL,1
LOO:
      CMP BYTE PTR [81H+SI],0DH
      JZ  NO_PAR
      MOV AL, [81H+SI]
      CMP AL,' '
      JZ  SPACE
      XOR DL,DL
      MOV [PATH2+DI] ,AL
```

[50] **Некоторое время назад мной в качестве тренировки были написаны несколько простых вирусов. Один из них до сих пор хранится где-то в архиве. Из этических соображений я не привожу его текст, используя лишь некоторые алгоритмы из него.**

```
        INC DI
        JMP SHORT L001
SPACE:
        OR  DL,DL ;если DL=0, тогда первый параметр закончился
        JZ  NO_PAR
L001:
        INC SI
        JMP SHORT LOO
NO_PAR:
        OR  SI,SI          ;был ли параметр
        JNZ CONT
;сообщение, затем выходим
        MOV DX,OFFSET TEXT1
        MOV AH,9
        INT 21H
        JMP _END
CONT:
;открываем файл, на который посадим сторожа
        LEA DX,PATH2
        MOV AH,3DH
        MOV AL,2
        INT 21H
        MOV BX,AX
        MOV HANDL2,BX
;найдем длину
        XOR DX,DX
        XOR CX,CX
        MOV AH,42H
        MOV AL,2
        INT 21H
;LEN-->AX
;проверка на длину
;правильнее было бы проверять длину файла + длина сторожа
/здесь мы предполагаем, что сторож не длиннее 1.5 Кб
        MOV LEN2,AX
        CMP AX,64000
        JNA _NO
        JMP _END
_NO:
;открываем для чтения "сторож"
        LEA DX,PATH1
        MOV AH,3DH
        MOV AL,0
        INT 21H
```

```
;читаем в буфер
        MOV BX,AX
        MOV AH,3FH
        LEA DX,BUF
        MOV CX,2000  ;полагаем, что 2000 байт хватит
        INT 21H
        MOV LEN1,AX
;закроем
        MOV AH,3EH
        INT 21H
;пишем в конец файла
        MOV BX,HANDL2
        MOV AH,40H
        LEA DX,BUF
        MOV CX,LEN1
        INT 21H
;на начало файла
        XOR CX,CX
        XOR DX,DX
        MOV AH,42H
        MOV AL,0
        INT 21H
;читаем 3 байта
        LEA DX,BUF
        MOV CX,3
        MOV AH,3FH
        INT 21H
;в нужное место
        MOV CX,0
        MOV DX,LEN2
        MOV AL,0
        MOV AH,42H
        INT 21H
;пишем
        LEA DX,BUF
        MOV CX,3
        MOV AH,40H
        INT 21H
;опять на начало
        XOR CX,CX
        XOR DX,DX
        MOV AH,42H
        MOV AL,0
        INT 21H
```

```
;заполним буфер, команда JMP ADR
        MOV BYTE PTR BUF,0E9H
        MOV AX,LEN2
        MOV WORD PTR BUF+1,AX
; пишем три байта
        MOV AH,40H
        LEA DX,BUF
        MOV CX,3
        INT 21H
;закроем
        MOV AH,3EH
        INT 21H
        LEA DX,TEXT2
        MOV AH,9
        INT 21H
_END:
        RETN
PATH1   DB 'PO.COM',0   ;имя сторожа
PATH2   DB 80 DUP(0)    ;имя COM-программы
TEXT1   DB 'Нет параметра',13,10, '$'
TEXT2   DB 'Сторож установлен',13,10,'$'
HANDL2 DW ? ;дескриптор файла
;длины файлов
LEN1     DW ?
LEN2     DW ?
BUF      DB 2000 DUP(?)
CODE ENDS
        END BEGIN
```

*Рис. 18.3. Программа установки сторожа на **COM-программу**.*

```
;программа удаления сторожа с COM-файла
CODE SEGMENT
        ASSUME CS:CODE, DS:CODE
        ORG 100H
BEGIN:
;блок анализа командной строки
        XOR SI,SI
        XOR DI,DI
        MOV DL,1
LOO:
        CMP  BYTE  PTR  [81H+SI],0DH
        JZ   NO_PAR
        MOV AL,[81H+SI]
```

```
        CMP AL,' '
        JZ   SPACE
        XOR DL,DL
        MOV [PATH1+DI],AL
        INC DI
        JMP  SHORT  L001
SPACE:
        OR   DL,DL  ;если DL=0,  тогда первый параметр закончился
        JZ   NO_PAR
L001:
        INC SI
        JMP SHORT LOO
NO_PAR:
        OR   SI,SI          ;был ли параметр
        JNZ CONT
;сообщение, затем выходим
        MOV DX,OFFSET TEXT1
        MOV AH,9
        INT 21H
        JMP _END
CONT:
;открываем файл, с которого удалим сторожа
        LEA DX,PATH1
        MOV AH,3DH
        MOV AL,2
        INT 21H
        MOV BX,AX
;найдем конец файла  (без сторожа)
        MOV AH,42H
        MOV AL,0
        XOR CX,CX
        MOV DX,1
        INT 21H
        MOV AH,3FH
        LEA DX,BUF
        MOV CX,2
        INT 21H
        MOV DX,WORD PTR BUF   ;в DX длина файла
        MOV LEN,DX
;теперь перейдем на конец
        MOV AH,42H
        MOV AL,0
        XOR CX,CX
        INT 21H
```

```
;прочесть 3 байта
        MOV AH,3FH
        LEA DX,BUF
        MOV CX,3
        INT 21H
;снова на начало
        MOV AH,42H
        MOV AL,0
        XOR CX,CX
        XOR DX,DX
        INT 21H
;пишем три байта
        MOV AH,40H
        LEA DX,BUF
        MOV CX,3
        INT 21H
;теперь читаем в буфер
;но вначале назад
        MOV AH,42H
        MOV AL,0
        XOR CX,CX
        XOR DX,DX
        INT 21H
;чтение в буфер
        MOV CX,LEN
        LEA DX,BUF
        MOV AH,3FH
        INT 21H
;закрытие файла
        MOV AH,3EH
        INT 21H
;создать файл (переписать старый)
        MOV AH,3CH
        LEA DX,PATH1
        MOV CX,0
        INT 21H
;записать в него буфер
        MOV AH,40H
        LEA DX,BUF
        MOV CX,LEN ;длина файла без сторожа
        INT 21H
;закрыть
        MOV AH,3EH
        INT 21H
        LEA DX,TEXT2
```

```
        MOV AH,9
        INT 21H
_END:
        RETN
PATH1   DB 80 DUP(O)     ;имя COM-программы
TEXT1   DB 'Нет параметра',13,10, '$'
TEXT2   DB 'Сторож удален',13,10, '$'
;длины файлов
LEN     DW ?
BUF     DB 64000 DUP(O)
CODE ENDS
        END BEGIN
```

Рис. 18.4. Программа удаления сторожа с COM-программы.

```
;программа PO.ASM
.286
CODE SEGMENT
ASSUME CS:CODE
ORG OOH
BEGIN:
;здесь три первых байта
DB 3 DUP(?)
;сохранить регистры
        PUSH  AX
        PUSH  DX
        PUSH  SI
        PUSH  DI
;восстановить три первых байта
        MOV   DI,100H
        MOV   SI, [DI+1]
        MOV   DX,SI   ;еще понадобится
        ADD   SI,100H
        MOV   AL, [SI]
        MOV   [DI],AL
        MOV   AL, [SI]+1
        MOV   [DI]+1,AL
        MOV   AL, [SI]+2
        MOV   [DI]+2,AL
;выполнить нужные действия
;например, проверка контрольной суммы
;**********************************
;**********************************
;**********************************
```

```
;вывод строки
        ADD    DX,100H
        LEA    AX,TEXT1
        ADD    DX,AX
        MOV    AH,9
        INT    21H
        MOV    AH,0
        INT    16H
;передать управление основной программе
;в начале восстановим регистры
        POP    DI
        POP    SI
        POP    DX
        POP    AX
        PUSH 100H ;команды 286-го процессора, MOV AX,100H/PUSH AX
        RETN
TEXT1 DB "Сторож сделал свое дело. Сторож может уходить.",13,10
        DB "Жмите любую клавишу.",13,10,"$"
_END:
CODE ENDS
        END BEGIN
```

Рис. 18.5. Программа сторож (каркас).

Далее приводится краткое пояснение вышеприведенных программ.

1. Первая программа, получая в командной строке имя COM-программы, подсоединяет к ней PO.COM. Сохраняет в первых трех байтах сторожа первые три байта самой программы. В начале основной программы ставится JMP ADR, где ADR - смещение сторожа. При этом уясните себе, что это не начало сторожа. Это начало плюс 3 байта. В этих трех байтах будет храниться начало программы.
2. Наш сторож не выполняет никаких действий, кроме вывода строки. Самое главное - разберитесь в адресации. Адрес любой строки здесь будет складываться из 100H,длины основной программы, смещения относительно начала программы "сторож".
3. Программа удаления "сторожа" делает действия, обратные программе на Рис. 18.3. Т.е. восстанавливает первые три байта и удаляет лишний код.

А теперь я предлагаю Вам, дорогой читатель, на основе наших программ написать свои программы, дополнительно выполняющие следующие действия:

1. Ваш сторож должен проверять контрольную сумму программы и выдавать сообщение, если контрольная сумма изменилась. В этом случае следует предусмотреть место, где эта контрольная сумма будет храниться.
2. Усовершенствуйте программу установки так, чтобы она проверяла возможную ситуацию, когда длина программы и "сторожа" превосходит 64 Кб.
3. Наконец предусмотрите проверку того, установлен сторож или нет. Простейший вариант: в конце программы "сторож" поместите строку, наличие которой потом будете проверять.

Глава 19. Проблемы компьютерной безопасности.

Бди!

Козьма Прутков.

Некоторые соображения, приведенные в данной главе, теряют свою актуальность в связи с развитием новых технологий: многозадачные операционные системы, глобальные и локальные сети. Однако сама проблема безопасности стала еще более актуальной в связи с тем, что решать эту проблему в изменившейся обстановке стало еще сложнее.

I. Общие соображения по поводу безопасности компьютерных систем.

По-видимому, не требует особых доказательств необходимость уделять серьезное внимание вопросам компьютерной безопасности. Достаточно назвать несколько нашумевших дел в банковских компьютерных системах, случаев сбоев в военных компьютерах или историй с похождениями компьютерных вирусов. К сожалению, мы не сможем рассмотреть все вопросы, связанные с компьютерной безопасностью, ограничимся лишь краткой их характеристикой. Более подробно будет рассмотрена проблема защиты программного обеспечения от несанкционированного копирования (что непосредственно связано с темой книги).

Рассмотрим основные проблемы, возникающие в данной области, и их краткую характеристику.

1. Защита от случайной потери информации (некомпетентность пользователя, сбой оборудования, вирус и т.п.). Особенно остро эта проблема стоит для компьютеров с несколькими пользователями.

   Возможные решения: копирование информации, разграничение доступа как на программном, так и на административном уровне.

2. Защита конфиденциальной информации от несанкционированного доступа. Дело здесь не только в том, что информацию могут украсть. Она может попасться на глаза человеку, которому не следует о ней знать.

   Возможные решения: разграничение доступа (см. ниже), правовые средства.

3. Защита программного обеспечения от несанкционированного использования. Здесь скрывается сразу несколько проблем.

   а) Программа может быть скопирована для использования ее по назначению.

   б) Из программы могут быть извлечены отдельные алгоритмы или процедуры для использования в своих целях.

   в) В программу могут быть введены некоторые изменения, например, в целях выдачи ее за свой продукт.

   г) Наконец, могут быть взяты некоторые оригинальные идеи, положенные в основу написания программы.

Возможные решения: защита от копирования и дизассемблера, правовые средства.

4. Защита от ошибок, возможно, содержащихся в программе. Проблема действительно актуальна, если вспомнить, что компьютеры в настоящее время управляют военным оборудованием, космическими кораблями, атомными станциями и т.д. Возможные решения: тщательное тестирование компьютерных систем на предмет аварийных ситуаций, дублирование функций отдельных узлов компьютерных систем, так чтобы при выходе из строя отдельного узла другие взяли на себя его функции (актуально в компьютерных сетях).

Рассмотрим возможную модель безопасности компьютерной системы, основанную на разграничении доступа. Пусть имеется некоторое множество объектов. Под объектами мы будем понимать некоторые элементы компьютерной системы. Это может быть файл, содержащий некоторую информацию, программа, отдельные функции программы, подкаталог, раздел, компьютер или группа компьютеров, объединенная в сеть и т.д. В общем, все то, на что может распространяться запрет или разрешение на использование. Скажем, разрешается пользоваться некоторой программой, которая выдает определенную информацию, но запрещено изменять информацию. Здесь объектом является отдельная функция программы. Но и сама программа может быть запрещена для использования. Можно запретить пользоваться данным разделом диска и т.д.

Другим множеством является множество субъектов, для каждого из которых может быть разрешен или запрещен доступ к данному объекту. Это могут быть не только люди, но и какие-то программы, не являющиеся элементами данной системы. Вопрос заключается в том, как установить соответствие между элементами этих двух множеств.

Рассмотрим, например, следующий подход. Пусть задано N уровней привилегий. Каждому элементу того и другого множества присваивается свой уровень привилегий. Предположим что, некоторому объекту присвоен K-й уровень привилегий, а некоторому субъекту I-й. Тогда доступ субъекта к объекту возможен лишь при условии I=>K. Такой подход позволяет формализовать систему компьютерной безопасности. Реально, однако, сложность может возникнуть при присвоении элементам множества уровней привилегий. В частности, не должно быть такой ситуации, когда по нескольким элементам можно было бы воссоздать элемент с более высокой привилегией. Предположим, что для Вас закрыты некоторые функции программы - Вы не знаете пароля. Однако Вам предоставлена полная возможность использовать на компьютере все отладочные средства или даже скопировать программу к себе на дискету и поработать с ней на другом компьютере. Если у программы нет специальной защиты от дизассемблирования, то появляется опасность, что пароль скоро станет известен.

Есть смысл ввести понятие прочности какой-либо системы защиты. Например, за прочность можно принять отношение времени жизни предмета защиты к времени преодоления данной защиты. Защиту с прочностью большей единицы можно считать надежной. В случае если время жизни объекта велико по сравнению со временами преодоления любой из защит, следует периодически обновлять средства защиты (например, менять пароли). При этом период смены защиты должен быть меньше, чем время преодоления защиты. Другим критерием прочности защиты может служить отношение финансовых затрат на создание средств защиты к затратам на преодоление защиты. При этом необходимо еще учитывать и стоимость самого объекта защиты (или потери, возникающие при взломе защиты).

Реализация изложенной модели (как впрочем, и других моделей) невозможна без защиты программ от копирования.

Возможные защиты программ от копирования мы разделим на два класса: программные и прочие. На программных способах остановимся несколько позднее, вначале же перечислим способы из класса "прочие".

Юридический способ. Существующие законы о защите авторских прав в принципе могут защитить Ваши программные продукты от несанкционированного доступа. Однако здесь встает серьезная проблема идентификации вашей программы (или ее части). При написании программы следует учитывать данный момент. Обычно в таких случаях в программе оставляют специальные метки, наличие которых в загрузочном модуле будет служить доказательством идентичности программ.

Средства психологической защиты. Полезно дать объявление, что в данном программном продукте встроен механизм защиты вне зависимости от того, есть он на самом деле или нет. Это может создать у нарушителей чувство психологической неуверенности. Если же программа будет время от времени напоминать, кто ее легальный пользователь, то это чувство неуверенности может усилиться.

Преимущество легальных программных продуктов перед нелегальными. Если программный продукт сопровождается полной документацией, отпечатанной типографским способом, то это, как ни странно, может повлиять на несанкционированное его распространение. Легальные пользователи должны быть зарегистрированы и иметь явные преимущества над нелегальными (полную документацию, последние версии программы со скидкой, бесплатные консультации и т.д.).

II. Защита программ от копирования.

Основная идея защиты от копирования заключается в распознавании того, что программный продукт запущен нелегально. Это может быть запрос пароля. В этом случае неправильный ввод пароля и будет являться индикатором того, что произошла несанкционированная попытка доступа к программному продукту. Однако пароль легко узнать. Такая защита достаточно надежна, когда пользователь программного продукта сам заинтересован в его нераспространении, скажем, в том случае, когда защищается дорогостоящий программный продукт, способный дать значительный экономический эффект. Такой способ защиты можно назвать сознательным способом привязки к пользователю. Совсем недавно мне сообщили о возможности так сказать бессознательной привязки к пользователю. Этот способ довольно интересен, но, на мой взгляд, вряд ли надежен. Суть его заключается в том, что программа запоминает некоторые психологические особенности пользователя при наборе, скажем, некоторой ключевой фразы. Это может быть время набора фразы и слова, временные интервалы между словами, скорость набора некоторых букв и т.д. В результате может быть выведен некоторый результирующий коэффициент, точнее, интервал. Попадания в данный интервал и будет означать идентификацию пользователя.

Другой способ защиты — это привязка к ключевой дискете. Идея его заключается в том, что создается некопируемая метка. Такая метка не переносится на другую дискету, по крайней мере при стандартном копировании. Наиболее простым способом является нанесение на дискету физического повреждения. Часто такое повреждение

наносится лазером. В этом случае дефектным может оказаться всего один сектор. Программа, в свою очередь, проверяет наличие такой метки на дискете, и в случае отсутствия последней делается вывод о несанкционированном запуске программы. Еще одним способ нанесения метки является нестандартное форматирование. Такую метку называют магнитной. Наиболее часто встречаются следующие варианты:

1. Вынос метки за пределы стандартного поля копирования;
2. Нестандартная разметка дорожки;
3. Привязка к временным параметрам;
4. Комбинированные методы.

На Рис. 19.1 представлены программы установки метки на дорожку 41 (а) и проверки наличия такой метки на дискете (б).

```
CODE  SEGMENT
      ASSUME CS:CODE
      ORG 100H
BEGIN:
      XOR CL,CL
      XOR DX,DX     ;диск A,  головка 0
      MOV CH,41     ;дорожка 41
      MOV AL, 9     ;число секторов
      LEA BX,BUF    ;описатель секторов
      MOV AH,5      ;форматировать
      INT 13H       ;дорожку
      JC  KON
;если форматирование прошло успешно,
;заполняем сектор 1 из буфера BUFFER
      XOR DX,DX     ;диск и головка
      MOV CH,41     ;дорожка
      MOV CL, 1     ;сектор 1
      MOV AL,1      ;один сектор
      MOV AH,3      ;писать
      LEA BX,BUFFER
      INT 13H
      JC  KON
      LEA DX,MES
      MOV AH, 9
      INT 21H    ;сообщение об успешной установке метки
KON:
      RET
BUF   DB 41,0,1,2
      DB 41,0,2,2
      DB 41,0,3,2
      DB 41,0,4,2
      DB 41,0,5,2
```

```
        DB 41,0,6,2
        DB 41,0,7,2
        DB 41,0,8,2
        DB 41,0,9,2
BUFFER DB 512 DUP(12)
MES   DB 'На дискете установлена метка!', 13,10, '$'
CODE ENDS
        END  BEGIN
```

(а) Программа установки метки на дискету. Выдается сообщение, если установка метки прошла успешно.

```
CODE  SEGMENT
      ASSUME CS:CODE
      ORG 100H
BEGIN:
;вначале проверяем, является данная дискета на 360 Кб или
;нет
;для этого читаем BOOT-сектор, проверяем слово по смещению
;13H
;см. глава 14
       XOR AL,AL
       MOV CX,1
       XOR DX,DX
       LEA BX,BUF
       PUSH BX
       INT 25H
       POP AX       ;лишнее слово из стека
       POP BX
       JNC  NO_KON
       JMP  KON
NO_KON:
       CMP WORD PTR [BX]+13H,720     ;количество секторов 720 ?
       JNZ KON
;здесь проверяем метку
       XOR  DX,DX                ;диск и головка
       MOV CH,41                 ;дорожка
       MOV CL,1                  ;сектор
       MOV AL,1                  ;один сектор
       LEA  BX,BUF               ;куда читать
       MOV AH,2                  ;читать
       INT  13H
       JC   KON
       CMP  BYTE PTR [BX+1],12 ;проверяем один байт (хотя
```

```
;желательно бы все
     JNZ  KON
     LEA  DX,MES
     MOV  AH,9
     INT  21H
KON:
     RET
MES  DB  'КЛЮЧЕВАЯ ДИСКЕТА! ',13,10,'$'
BUF  DB 512 DUP(?)
CODE ENDS
     END  BEGIN
```

(б) Программа проверки наличия метки на дискете. Выдается сообщение, если дискета ключевая.

Рис. 19.1. Пример установки метки вне стандартного поля копирования.

Меняя описатель и таблицу параметров (см. главу 14) дискеты, можно произвести нестандартное форматирование дорожки. Вы можете менять нумерацию секторов, длину межсекторного интервала, длину сектора и т.д. Для правильного чтения такой дорожки таблица параметров дискеты должна быть такой же, как при форматировании. Можно произвести форматирование дорожки с большим числом секторов, а затем произвести запись в сектор с большой длиной [10, 14, 17][51].

При установке физической метки (прожигание, прокол) следует вначале исследовать характер повреждения (количество испорченных секторов и их координаты). Затем характер повреждения должен быть учтен при защите программы.

Защита от копирования нужна и инсталляционной дискете. Такие дискеты, как правило, содержат программу установки пакета на жесткий диск. Причем установка может производиться лишь энное число раз. По достижении данного числа данные на дискете безнадежно портятся.[52]

Привязка к жесткому диску обладает своей спецификой. Как правило, пакет для постоянной работы устанавливается на винчестер. Дискета же используется в основном в качестве инсталляционной. Установка на дискете ключа, для того чтобы при каждом запуске пакета требовалось вставить эту дискету в дисковод, является, на мой взгляд, не слишком эффективным средством защиты. На жесткий диск невозможно поставить физическую метку. Кроме того, копирование ЖМД один к одному весьма проблематично.

При установке пакета на жесткий диск возможна привязка к положению его файлов на диске (физические или логические координаты). Может, однако, возникнуть проблема при реорганизации диска, например, с помощью утилиты SPEEDISK. Правда, файл

---

[51] Подробнее о структуре дорожек на гибком диске см. [15], а также Приложение 9.

[52] Большинство программ в настоящее время инсталлируются с лазерного диска. Изменить данные на нем невозможно обычном способом, но можно, например, потребовать, чтобы диск всегда был в дисководе для успешной работы программы.

может быть помечен как неперемещаемый (READ ONLY). Однако атрибут файла может быть случайно изменен, что приведет к тому, что пакет не будет работать. В качестве примера, однако, я приведу программу, определяющую начальный кластер заданного файла (Рис. 19.2). Для определения начального кластера файла используется функция 52H (см. главу 14). Конечно, начальный кластер файла можно найти, поработав с FAT, но это гораздо сложнее. Данная программа определяет номер начального кластера файла с именем NONAME.COM и записывает его в него же по смещению OFSET.

```
CODE  SEGMENT
      ASSUME CS:CODE, DS:CODE
      ORG 100H
BEG:
;открыть файл
      MOV AX,3D02H
      LEA  DX,PATH
      INT  21H
      JC   _END
      PUSH  AX
;байт в таблице дескрипторов файлов равен
;номеру блока описания файла
      MOV SI,AX
      MOV DL,ES: [SI+18H]
      XOR DH,DH ;номер блока в DX
;адрес списка списков
      MOV AH,52H
      INT 21H
;указатель на системную таблицу файлов
      LES SI,ES:[BX+4]  ;в ES:SI
;ищем блок описания файла
      CMP DX,ES:[SI+4]
      JB  YES
      SUB DX,ES:[SI+4]
      LES SI,ES:[SI]   ;второй блок
;номер кластера
YES:
      MOV AX,59 /размер блока
      MUL DL
      ADD SI,AX
      MOV AX,ES:[SI+6+0BH]  ;номер кластера
      MOV CLAST,AX
;записываем номер по смещению OFSET, вначале сдвиг,
      POP BX
      MOV AX,4200H
      XOR CX,CX
```

```
        MOV  DX,OFSET
        INT  21H
;теперь запись
        MOV  AH,40H
        MOV  CX,2
        LEA  DX,CLAST
        INT  21H
;закрыть
        MOV  AH,3EH
        INT  21H
_END:
        RET
;имя программы
PATH DB  'NONAME.COM',0
;смещение, куда должен быть помещен номер кластера
OFSET DW 15
;кластер
CLAST DW ?
CODE ENDS
        END BEG
```

*Рис. 19.2. **Определение** начального кластера файла.*

Для винчестера возможна длинная запись (функция OBH) - записывается 516 байтов (512+4). В последние четыре байта можно записать идентификационную метку. Содержимое сектора будет меняться, а метка останется (см. Рис. 19.3).

```
CODE  SEGMENT
        ASSUME  CS:CODE
        ORG 100H
BEGIN:
;читаем: головка 2, дорожка 3, сектор 4
        MOV AH,AH
        MOV AL,1
        MOV DH,2
        MOV CL,4
        MOV CH,3
        MOV DL,80H
        MOV BX,OFFSET BUF
        INT  13H
        JC   ERR_D
;пишем 516 байт
        MOV  AH,0BH
        MOV  AL,1
```

```
        MOV DH,2
        MOV CL,4
        MOV CH,3
        MOV DL,80H
        MOV BX,OFFSET BUF
        INT  13H
        JC   ERR_D
        MOV  DX,OFFSET MES1
        JMP  SHORT WR
ERR_D:
        MOV  DX,OFFSET MES2
WR:
        MOV AH, 9
        INT 21H
        RET
MES1  DB  'Метка установлена на жесткий диск.', 13,10, '$'
MES2  DB  'При установке метки на жесткий диск произошла'
      DB  'ошибка.',13,10, '$'
BUF   DB 512 DUP(?)
        DB'1234'
CODE  ENDS
        END   BEGIN
```

(а) Установка метки на жестком диске.

```
CODE   SEGMENT
        ASSUME  CS:CODE
        ORG  100H
BEGIN:
;читаем: головка 2, дорожка 3, сектор 4
        MOV  AH,0AH
        MOV  AL,1
        MOV  DL,80H
        MOV  DH,2
        MOV CH, 3
        MOV  CL,4
        MOV  BX,OFFSET  BUFFER
        INT  13H
        JC  NO_M
;проверка последних 4 байт
        CMP  BYTE  PTR  ES: [BX]+512, '1'
        JNZ NO_M
        CMP  BYTE  PTR  ES: [BX]+513, '2'
        JNZ  NO_M
```

```
        CMP  BYTE  PTR  ES: [BX]+514,'3'
        JNZ NO_M
        CMP  BYTE  PTR  ES: [BX]+515,'4'
        JNZ  NO_M
        MOV  DX,OFFSET MES2
        JMP  SHORT WR
NO_M:
        MOV  DX,OFFSET MES1
WR:
        MOV AH, 9
        INT  21H
        RET
BUFFER    DB  512  DUP(?)
MET       DB  4  DUP(?)
MES1       DB  'Метка на  жестком  диске  не  обнаружена.'
           DB  13,10,'$'
MES2       DB  'Метка на  жестком  диске  обнаружена.',13,10,'$'
CODE  ENDS
     END  BEGIN
```

(б) Проверка метки на жестком диске.

Рис. 19.3. Пример установки (а) и проверки (б) метки на жестком диске.

Информация, необходимая при идентификации пакета, может быть помещена в BOOT-сектор или PARTITION TABLE. Однако это ненадежный способ, и прибегать к нему следует лишь в простых ситуациях. Дело в том, что содержимое этих секторов может измениться при переустановке системы. Некоторые же антивирусные программы могут принять постороннюю информацию в секторе за наличие вируса. Пакет может быть привязан к BAD-кластерам, которые проставляются во время установки.

Привязка к винчестеру может сочетаться с проверкой некоторых индивидуальных параметров компьютера. К таковым относятся:

- серийный номер микропроцессора, микросхемы ПЗУ и дата изготовления;
- аппаратная конфигурация данного компьютера;
- содержание ПЗУ (контрольная сумма);
- частотные и временные характеристики отдельных узлов.

III. Защита от дизассемблеров.

Если взломщику удастся узнать механизм защиты, то проблема переноса пакета на другой компьютер будет решена. Основным средством взлома является отладчик - программа, позволяющая контролировать выполнение других программ (пошаговое выполнение, останов в нужном месте и возврат в отладчик, просмотр кода и т.п.). Как противодействовать этому? Общий ответ гласит: программа должна узнать (почувствовать), когда она будет работать под отладчиком.

Использование прерываний с номерами 1 и 3. Эти вектора перенаправляют на себя практически все отладчики.[53] Первый вектор служит для пошагового выполнения программы. Если флаг трассировки взведен, то после каждой команды происходит обращение по этому вектору (INT 1). Вектор 3 служит для установки в программе точек останова. Команда INT 3 (в отличие от других INT) занимает всего один байт. Используя изложенные факты, Ваша программа может заставить отладчик идти по ложному пути или вообще отказаться работать. Например, незаметно можно переустановить вектор 1, что даст мгновенный результат. Аналогично можно действовать и с вектором 3, хотя некоторые отладчики восстанавливают это вектор после исполнения каждой команды. Представленный ниже код будет срабатывать довольно эффективно, особенно если его "разбавить" другими командами.

```
        XOR AX,AX
        MOV ES,AX
        MOV BX,4
        MOV ES:[BX+2],1234H
```

Проверка на наличие в коде точек останова может проверяться по контрольной сумме отдельных участков. Если она изменилась, то можно делать вывод о модификации кода точками останова. Возможен вариант проверки того, куда направлены эти векторы - в системную область или нет.

Использование очереди команд. При работе в отладчике очередь команд обнуляется после каждой команды, тогда как в реальной ситуации это не так. Рассмотрим следующий фрагмент:

```
        MOV BX,OFFSET MET2
        JMP MET1
MET1:
        MOV BYTE PTR CS:[BX],0C3H
        .
        .
        .
MET2:
```

В реальной ситуации, если команда с меткой MET2 попадает в буфер команд вместе с командой ее модификации, то никакого эффекта это не произведет. Под отладчиком же она будет модифицирована.

Преобразование кода программы в процессе ее выполнения. Данный метод весьма продуктивен, если условием модификации кода программы поставить наличие или отсутствие отладчика. Обычно реальный исполняемый участок программы получается путем некоторых арифметических операций с определенными данными, также храня-

---

[53] Речь в данном случае идет, разумеется, об операционной системе MS DOS. Отладка программ, написанных для Windows, имеет свою специфику.

щимися в коде программы. Ключ к расшифровке при этом может браться с защищенной дискеты. Некоторые фрагменты программы можно помещать в область стека, тогда, если отладчик будет использовать стек программы, фрагменты будут испорчены.

Противодействие общению с пользователем. Типичным примером противодействия общению с пользователем является отключение прерывания клавиатуры на уровне контроллера прерываний (на уровне микропроцессора это не удастся, т.к. отладчики следят за этим).

```
MOV AL,2
OUT 21H,AL
```

Некоторые отладчики, однако, на каждом шаге восстанавливают порт, и такой «фокус» с ними не проходит.

Учет временных зависимостей работы программы. Используя прерывание 1CH или 8H, можно проверять время работы различных участков программы. Естественно, под отладчиком эти участки будут работать намного медленнее.

Этот способ можно назвать "изматыванием противника". Можно написать целые фрагменты, которые будут иметь самый "серьезный" вид, но ничего не значить. Значащие команды можно разбавлять незначащими. Интенсивный обмен информацией с памятью, беспрерывный вызов процедур может легко сбить столку. Конечно, данный подход следует сочетать с другими.

Регистрация работы отладчика. Мы уже говорили о регистрации работы отладчика путем проверки векторов 1 и 3. Теперь поговорим о других способах.

а) Проверка содержимого регистров. При входе в программу через обычный DOS'овский загрузчик почти все регистры общего значения несут содержательную нагрузку, чаще всего имея не нулевое значение. Большинство же отладчиков при входе в программу обнуляют эти регистры. Это может служить хорошим критерием по проверке наличия отладчика.

б) В PSP по смещению 2EH находится область, куда DOS помещает указатель на стек при обращении к системным функциям. Выполните какую-нибудь команду DOS, а потом сравните:

```
MOV AX,SS
CMP AX,ES:[30H] ;ES на PSP
```

Отладчик помещает в ячейку указатель на свой стек, поэтому содержимое AX и ES:[30H] не должно совпадать.

в) В дочерней программе PSP по смещению 16H хранится сегментный адрес предка. Для COMMAND.COM это всегда ссылка на свой PSP, даже если запущена копия командного микропроцессора.

```
MOV AX,ES:[16H]
MOV ES,AX
CMP AX,ES:[16H]
```

Если содержимое AX не равно ES:[16H], то работает отладчик. Разумеется, в данном случае за отладчик будет принята любая программа, запускающая Вашу.

Другой опасностью для защищенных программ является трассировка прерываний. Под трассировкой прерывания в данном случае имеется в виду возможность отслеживания входной и выходной информации этого прерывания при работе какой-нибудь программы. Зная такую информацию, можно, например, судить о том, какую программа получает информацию с ключевой дискеты, и снять защиту. В основном это касается двух прерываний: 21H и 13H. Здесь можно дать несколько рекомендаций:

1. Отслеживание перехвата прерываний после запуска программы. Программа может расценивать такой перехват как попытку несанкционированного доступа к ней.
2. Вызов прерываний обходным путем. Определение значения векторов, которые они имели сразу после загрузки системы. Например, функция 13H прерывания 2FH позволяет определить адрес вызова прерывания 13H в BIOS. Косвенный вызов прерывания 21H через функцию 5DH.

Следующий фрагмент демонстрирует использование прерывания 2FH.

```
MOV AH,13H
INT  2FH
PUSH  DS
PUSH  BX
INT  2FH
POP  BX
POP  DS
```

Поясню работу данного фрагмента. Функция 13H одновременно и устанавливает вектор, и возвращает данные. Перед вызовом DS:BX должен содержать новое значение вектора 13H. Возвращается же в тех же регистрах адрес прерывания 13H в BIOS. Наш фрагмент сразу устанавливает вектор. Если какой-либо программой этот вектор был раньше перехвачен, она будет исключена из цепочки прерываний. Можно поступить по-другому - не трогать вектор 13H, а использовать вместо него другой свободный вектор.

Ниже (Рис. 19.4) представлена программа, где косвенно вызывается функция 9 DOS через функцию 5DH.

```
CODE SEGMENT
       ASSUME  CS:CODE,DS:CODE
       ORG  100H
BEGIN:
       MOV  DX,DS
       MOV  _DS,DX
       MOV  _AX,0900H
       MOV  _DX,OFFSET MES
       MOV  DX,OFFSET _AX
       MOV  AX,5D00H
```

```
        INT  21H
        RET
MES  DB 'Косвенный вызов функции 09, прерывания 21H',13,10,'$'
;содержимое регистров
_AX   DW  ?
_BX   DW  ?
_CX   DW  ?
_DX   DW  ?
_SI   DW  ?
_DI   DW  ?
_DS   DW  ?
_ES   DW  ?
CODE  ENDS
        END  BEGIN
```

Рис. 19.4. Косвенный вызов функций DOS.

IV. Защита жестких дисков от несанкционированного доступа.

Здесь приводится схема (достаточно простая) защиты жестких дисков, разделов на нем и отдельных файлов от несанкционированного доступа. Поскольку MS DOS не содержит в себе средства разделения доступа, то все такие защиты действуют по вирусному принципу, т.е. нелегально. Проблема такой защиты возникает особенно часто в учебных заведениях, где один и тот же компьютер используют несколько человек. Рассмотрим лишь общую схему без текстов программ.

1. Защиту жесткого диска осуществляет программа, заменяющая главную загрузочную запись. Сама же загрузочная запись, а также BOOT-сектор разделов хранятся на жестком диске в зашифрованном виде. Запуск операционной системы с дискеты не помогает, так как в этом случае жесткий диск для операционной системы перестает существовать. Стандартные методы восстановления загрузочных секторов не работают в этом случае, т.к. неизвестно, где начинаются разделы.
2. Вход на жесткий диск осуществляется по паролю. Пароль определяет, какие разделы, каталоги, файлы, а также операции будут доступны пользователю. Если пароль введен правильно, то защищающая программа обеспечивает запуск операционной системы с жесткого диска. При этом в памяти остается некоторый резидентный модуль, который и осуществляет разграничение доступа.
3. Резидентный модуль должен обеспечить правильность работы операционной системы и других программ. В частности, должно быть сымитировано правильное содержимое всех загрузочных секторов, однако заблокирована возможность снятия с них копии обычными способами (через прерывание 13H).
4. Резидентный модуль осуществляет разграничение доступа к разделам жесткого диска, каталогам и отдельным файлам, а также блокирование отдельных опера-

ций с гибкими дисками (например, запуск и копирование программы с них) посредством контроля над прерываниями 13H, 21H, 25H, 26H. Блокирование операций с гибкими дисками не позволяет запускать какие-либо программы, которые могли бы помочь снять защиту.

5. Резидентный модуль осуществляет свою собственную защиту в памяти.
6. Для полноты в данную схему можно было бы добавить также регистрацию попыток несанкционированных операций пользователя, который идентифицируется своим паролем. Это дало бы возможность оперативно реагировать на все попытки "взлома" системы сменой паролей, перераспределением доступа, применением к взломщику административных мер и т.д.

Данная схема в принципе проста и может быть усилена средствами административного воздействия. К сожалению, автору не довелось встретиться со средствами защиты, которые бы воплощали в себе все вышеописанные свойства [54], хотя отдельные моменты реализованы во многих системах защиты.

V. Защита в локальной сети.

В настоящее время нельзя обойти вопрос безопасности компьютеров, работающих в локальной сети. Здесь в первую очередь следует разделить информациьо, хранящуюся на локальных дисках и информацию, хранящуюся на сетевых дисках (сетевая информация). К первой информации всецело относится все то, что было сказано выше. Сетевая информация защищена средствами сетевой операционной системы. Мы рассмотрим средства операционной системы Novel Netware вер. 3.12 и выше.

Здесь, как было сказано выше, можно также ввести понятие объекта. Все множество объектов можно разделить на два подмножества: подмножество файлов и каталогов, с одной стороны, и подмножество пользователей - точнее, имен, под которыми может зарегистрироваться пользователь. Безопасность компьютерной сети определяется взаимодействием прав и ограничений, накладываемых на элементы данных подмножеств.

ОС Novel Netware обеспечивает четыре уровня защиты. Первые два уровня связаны с наложением ограничений на пользователей, вторые два уровня – с ограничениями на каталоги и файлы.

Пользователи.

1-й уровень.

1. Ограничение паролем. Для работы в сети каждый пользователь должен зарегистрироваться. При регистрации пользователь получает имя, которое определяет уровень его привилегий. Данное имя может быть защищено паролем. В этом случае получение данных привилегий может быть осуществлено лишь при правильном указании пароля.
2. Ограничение времени работы. Для каждого имени можно указать время (часы, дни), в которые пользователь может работать (зарегистрироваться).
3. Ограничение дискового пространства. Для данного имени может быть указан объем дискового пространства (на каждый том), которое может быть использовано.

[54] Речь, разумеется, идет о защите в среде MS DOS.

4. Регистрация нарушителей. При попытках зарегистрироваться с неправильным паролем бюджет для данного имени может быть закрыт.
5. Возможность работать только на конкретных машинах. Каждая машина имеет свой сетевой адрес, который задается на сетевой плате. Для каждого имени можно указать те машины, где можно зарегистрироваться.
6. Процедура регистрации. Для каждого имени может быть определена процедура регистрации - те команды, которые выполняются при регистрации под данным именем. В частности, в процедуре регистрации определяются сетевые диски.

2-й уровень.

Назначение опекуна в каталоги и файлы.

При назначении опекуна в каталоги задается маска того, что данный пользователь может делать с каталогом или файлом. Вот эти права:

1. Администраторское (права Supervisor'a- можно делать все с каталогом или файлом)
2. Запись
3. Изменение
4. Контроль доступа (изменение прав доступа к каталогу или файлу)
5. Поиск файла
6. Создание (только для каталогов)
7. Удаление
8. Чтение.

При назначении опекуна в каталог его можно лишить всех прав, тогда данный каталог для него будет полностью недоступен.

Каталоги и файлы.

3-й уровень.

Наследуемые права каталогов (или файлов). Каждый каталог получает свой набор прав, которые он предоставляет опекунам. Из прав каталога и прав опекуна складываются эффективные права, которые имеет данный пользователь по отношению к данному каталогу. Причем права каталога имеют приоритет над правами опекуна.

4-й уровень.

Атрибуты файла или каталога.

Каждый файл или каталог имеет набор атрибутов. Их гораздо больше, чем в файловой системе MS DOS. Эти атрибуты имеют самый высокий приоритет. Поясним это на примере. Если эффективные права опекуна данного файла позволяют ему его удалять, а атрибут файла запрещает удаление, то такой файл удалить нельзя.

Автор считает, и в этом он не одинок, что сетевая операционная система Novel более совершенна в смысле защиты информации, чем другие.[55]

[55] Хорошо известна такая шутка, впрочем, весьма близкая к действительности. Если Вы хотите выбрать надежную и удобную сеть - выбирайте сеть Novel или Unix, если Вы не хотите ни того, ни другого - выбирайте NT.

Глава 20. Микропроцессоры 8086, 80186, 80286, 80386, 80486, Pentium...

Так выпьем же за то, чтобы, как бы высоко мы ни поднимались, никогда бы не отрывались от коллектива!

Из кинофильма "Кавказская пленница"

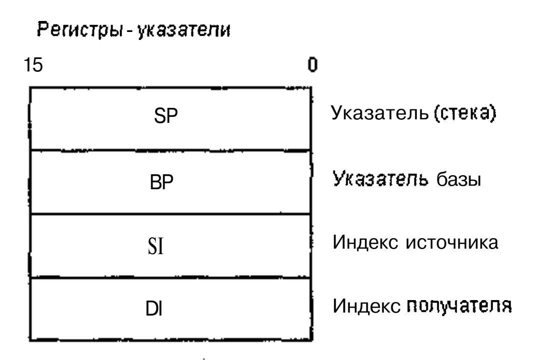
Как быстро летит время! Вот уже 386-е AT-ишки стали музейной редкостью. А стало быть, пора познакомиться с возможностями всего семейства микропроцессоров. Данная глава и содержит в себе сравнительный анализ того, как менялись возможности микропроцессоров семейства INTEL. Рассмотрение будет касаться в основном программной части (регистры, команды), т.е. того, что может пригодиться при программировании.

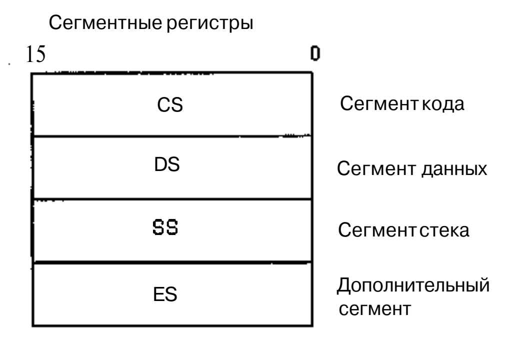
Первым полноправным членом семейства INTEL был процессор 8086 (1978). Этот микропроцессор имел 16-битную структура как внутри, так и при обмене с памятью. Через год появился микропроцессор 8088, который был полностью совместим с микропроцессором 8086, но работал с 8-битной шиной данных. Для программиста это было совершенно незаметно, так как выполнялись те же самые команды. Просто команда типа MOV MEM,AX выполнялась в таком микропроцессоре в два приема. Микропроцессор 8088 использовался с дешевыми микросхемами памяти, поэтому нашел широкое применение. Если теперь читатель попытается найти в настоящее время EXT-ишку, то скорее всего это будет компьютер на базе микропроцессора 8088.

В ряду микропроцессоров рассматриваемого семейства были три особые (можно сказать контрольные) точки: 8086 - начало семейства; 80286 - появление защищенного режима; 80386 - 32-битные регистры, виртуальный режим, страничная адресация.

I. Регистры.

МП 8086/8088.







Значение флагов:
0 (CF) - флажок переноса.
1 (1) - резерв.
2 (PF) - флажок приоритета.
3 (0) - резерв.
4 (AF) - флажок дополнительного переноса.
5 (0) - резерв.
6 (ZF) - флажок нуля.
7 (SF) - флажок знака.
8 (TF) - флажок трассировки.
9 (IF) - флажок разрешения прерываний.
10 (DF) - флажок направления.
11 (OF) - флажок переполнения.
12-15 - резерв (0).

МП 80186/80188.

Данных по этому микропроцессору у меня нет, но из общих соображений можно заключить, что система регистров здесь такая же, что и в 8086/8088.

МП 80286.

Основные регистры остались те же.

Рабочие регистры (80286)

	15	0	
AX	AL	AH	Аккумулятор
BX	BL	BH	База
CX	CL	CH	Счетчик
DX	DL	DH	Данные

Регистры - указатели (80286)

15 … 0	
SP	Указатель (стека)
BP	Указатель базы
SI	Индекс источника
DI	Индекс получателя

Сегментные регистры (80286)

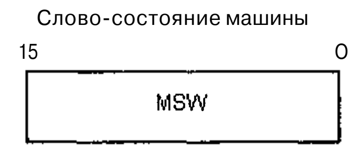
15 … 0	
CS	Сегмент кода
DS	Сегмент данных
SS	Сегмент стека
ES	Дополнительный сегмент

Программный счетчик (80286)

15 0

IP

Был добавлен регистр, касающийся защищенного режима.



Значение битов:

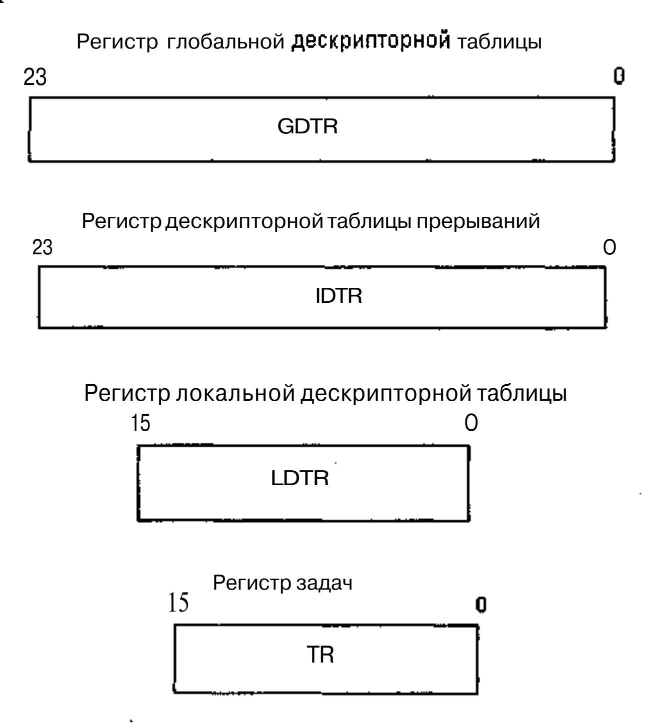
0 (PE) - бит защищенного режима. Установка данного бита в 1 разрешает защиту на уровне сегментов.

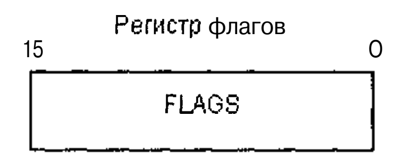
1 (MP) - бит наличия сопроцессора. Управляет функцией команды ожидания. Начиная с микропроцессора 80486 должен всегда быть равен 1.

2 (EM) - бит отсутствия сопроцессора. Если бит установлен в 1, то выполнение любой команды сопроцессора или команды WAIT вызывает особый случай (7).

3 (TS) - признак, что задача была переключена.

4-15 - резерв.





Добавилось (по сравнению с 8086) использование двух новых битов, касающихся работы в защищенном режиме.

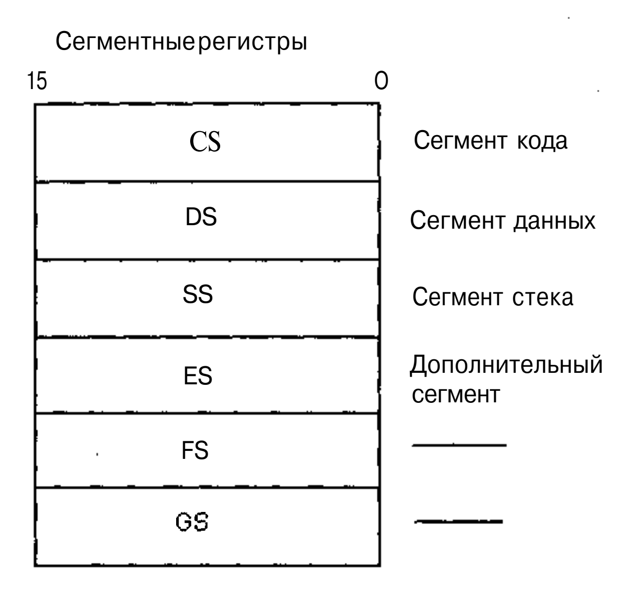
12-13(IOPL) - уровень привилегий.
14 (NT) - вложенная задача.

МП 80386.

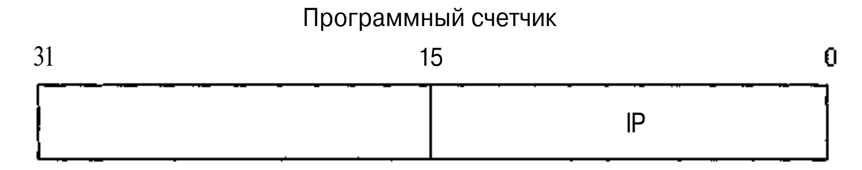
Большая часть регистров процессора стали 32-битными.







Добавилось два новых сегментных регистра.



Структура дескриптора для процессора 80386 осталась той же (см. главу 5), но старшие 16 бит, бывшие в резерве, теперь используются. В частности, там хранится теперь старший байт базового 32-битного адреса сегмента.

Регистры GDTR, LDTR, IDTR, TR имеют такие же размеры, как и у микропроцессора 80286.



Добавилось (по сравнению с 80286) использование двух новых битов.

16 - флаг итога - используется при отладке. Используется для временного запрещения особых случаев отладки.

17 - флаг виртуального режима - включает виртуальный режим.

Другие регистры.

Регистр управления

31	15	0
	CR0	

Младшие 16 бит регистра CRO называются словом состояния машины (см. микропроцессор 80286).

Биты:

0 - PE - включение защищенного режима,

1 - MP - управление сопроцессором,

2 - EM - эмуляция сопроцессора (бит отсутствия),

3 - TS - бит переключения задач.

31 - PG - включение управления страницами.

Т.о. добавился один новый бит (31-й).

Регистр CR1 - зарезервирован.

Регистр CR2 - содержит линейный адрес ошибки страницы.

Регистр CR3 - базовый адрес каталога страниц (биты 12-31).

Таким образом, добавились регистры CR1-CR3. Слово состояния машины превратилось в регистр CRO.

Регистры отладки (32-битные).

DRO-DR3 - четыре линейных точки останова. Формируемые программой адреса при выполнении каждой команды сравниваются с адресами данных регистров. При совпадении адресов генерируется сигнал особого случая отладки.

DR4-DR5 - зарезервировано.

DR6 - текущее состояние точек останова (текущее состояние отладки).

DR7 - регистр управления отладкой. Для каждой контрольной точки отводится поле (8 бит), позволяющее задать:

- регистрацию контрольной точки в одной или любой задаче;
- регистрацию контрольной точки при обращении к памяти для записи/считывания данных или для выборки команды;
- фиксацию контрольной точки в зависимости от размера операнда, к которому производится обращение.

Еще регистры (32-битные).

TR6,TR7 - (тест - регистры) регистры для работы с буфером ассоциативной связи кэш-памяти для трансляции линейных адресов в физические.

МП 80486.

Не перечисляя регистры 486-го микропроцессора, укажу лишь отличия его в этом плане от 386-го микропроцессора.

Регистр флажков дополнился новым флагом АС - 18-й. Флажок контроля выравнивания.

Регистр CRO также дополнился новыми флажками.

4 - ЕТ - тип расширения. Поддержка команд математического сопроцессора.

5 - NE - бит численной ошибки. Бит разрешения стандартного механизма численной ошибки.

6 - WP - бит защиты от записи. Защищает от записи страницы уровня пользователя от обращения режима супервизора.

7 - AM - запрещает или разрешает контроль выравнивания.

8 - NW - бит несквозной записи. Запрещение сквозной записи (1) может привести к появлению в кэш-памяти устаревших данных.

9 - CD - запрещение или разрешение внутренней кэш-памяти.

Регистр CR3.

Биты 12-31 - физический адрес каталога страниц.

Бит 2 - PWT - сквозная запись.

Бит 3 - PSD - запрещение кэширования страницы.

Кроме того, в состав микропроцессора 80486 фактически вошел арифметический сопроцессор, полностью совместимый с процессором 387.

МП PENTIUM.

В регистре флагов дополнительно использованы разряды 19-21.

21 – ID - флаг доступности команды идентификации (CPUID);

20 - VIP - виртуальный запрос прерывания;

19 - VIF - виртуальная версия флага IF для многозадачных систем.

Добавлен управляющий регистр CR4 с разрядами 0-6.

0 – VME - разрешение использования виртуального флага прерываний в режиме V86,

1 – PVI - разрешение использования виртуального флага прерываний в защищенном режиме,

2 – TSD - превращение инструкции RDTSC в привилегированную,

3 – DE - разрешение точек останова по обращению к портам ввода-вывода.

В регистре DR6 невозможно изменить значение разряда 12. Кроме того, в микропроцессоре Pentium увеличено количество так называемых тест-регистров. Их количество достигает теперь 12: TR1-TR12.

II. Команды реального режима.

МП 8086/8088.

Команды данного микропроцессора описаны в Главе 4 и Приложении 1, и здесь мы не будем повторяться.

МП 80186/80188.

О командах данного микропроцессора см. также главу 5.

BOUND REG,MEM - контроль попадания в диапазон.

В регистр помещается длина массива. В MEM находится нижнее значение индекса, в MEM+2 - верхнее значение. Если длина массива выходит за указанный диапазон, то генерируется прерывание 5.

ENTER N,M - образование стекового кадра для процедуры.

ENTER 16,3 - резервируется 16 байт для локальных переменных. Уровень вложенности 3.

Данная команда используется в основном в трансляторов языков высокого уровня.

LEAVE - удаление стекового кадра процедуры. Работает в паре с командой ENTER.

INS - ввод циклический. INSB, INSW, INSD. Работает как строковая команда.

OUTS - вывод циклический. OUTSB, OUTSW, OUTD. Работает как строковая команда.

PUSHA, POPA - поместить в стек и извлечь из него регистры DI, SI, BP, SP, BX, DX, CX, AX.

PUSH N, N - число.

Расширена команда знакового умножения.

Например, IMUL DX,CX,123 (BX*123->DX) (см. главу 5).

Расширена команда битового сдвига: SHL AX,N , где N> 1.

МП 80286.

Множество команд реального режима совпадает со множеством команд микропроцессора 80186.

МП 80386.

BSF REG,REG/MEM сканирование бита вперед. Команда сканирует второй операнд и заносит номер первого единичного бита в первый операнд. Если единички нет, то устанавливается бит Z.

BSR REG,REG/MEM сканирование бита назад. Аналогичная предыдущей.

BT REG,REG/MEM/N флажку CF передается бит из первого операнда за номером, определяемым вторым операндом.

BTC REG,REG/MEM/N инвертирует бит в первом операнде, номер которого определяется вторым операндом. Старый бит передается в CF.

BTR REG,REG/MEM/N сбрасывает бит в первом операнде, номер которого определяется вторым операндом. Старый бит передается в CF.

BTR REG,REG/MEM/N устанавливается бит в первом операнде, номер которого определяется вторым операндом. Старый бит передается в CF.

CWDE - преобразование слова в двойное слово.

CDQ - преобразование двойного слова в четвертное слово.

SHLD REG/MEM,MEM,N (SHRD REG/MEM,MEM,N) сдвиг двойной (вправо, влево). Команда двойного сдвига перемещает набор бит первого операнда влево или вправо на число разрядов, определяемым третьим операндом. Освобождающиеся биты заполняются из второго операнда.

SETCC - установка байта по условию. Указанный байт операнда устанавливается в 1 в зависимости от условия. Условие аналогично условиям переходов.

Множество команд расширено по сравнению с командами микропроцессора 80286:

Во-первых, за счет использования в командах 32-битных регистров. Например, ADD EAX,234567H.

Во-вторых, за счет использования суффикса D. Например, CMPSD - сравнение строковых двойных строк, IRETD - возврат из прерывания в 32-битном режиме, MOVSD - пересылка строки из двойных слов и т.д.

МП 80486.

Новые команды: BSWAP, XADD, CMPXCHG.

BSWAP изменяет порядок байт в 32-битном регистровом операнде.

XADD MEM,REG - источник заменяется получателем, а получатель суммой источника и получателя.

CMPXCHG MEM,AK,REG (AK - аккумулятор) - если значение операнда получателя и аккумулятора равны, операнд-получатель заменяется операндом-источником. В противном случае операнд-получатель загружается в аккумулятор.

МП PENTIUM.

Добавлены следующие команды:

CMPXCHG8B, CPUID, RDTSC, RDMSR, WRMSR, RDTSC и стандартные команды загрузки и выгрузки регистра CR4.

Рассмотрим эти команды.

CMPXCHG8B REG/MEM,REG - сравнение и обмен восьмибайтовых величин.

CPUID - получение информации о процессоре.

RDMSR REG/MEM - чтение регистров (TR1-TR12).

WRMSR REG/MEM - запись регистров (TR1-TR12).

RDTSC - чтение счётчика тактов.

III. Команды защищенного режима.

МП 8086/8088.

Команды защищенного режима отсутствуют.

МП 80186/80188.

Команды защищенного режима отсутствуют.

МП 80286.

Данные команды появились в процессоре для обслуживания защищенного режима.

ARPL - коррекция в селекторе уровня привилегий инициатора запроса.

LGDT - загружает GDT из памяти. При загрузке передается шесть байт. Первые пять байт загружаются в регистр GDT, шестой байт игнорируется (загружается в 386-м процессоре). Используется при инициализации. LGDT MEM.

SGDT - передает содержимое регистра GDT в адресуемую область памяти. Команда используется в системных отладчиках.

SGDT MEM.

LLDT - загружает операнд-слово из регистра или памяти в регистр LDT. Данное слово будет являться селектором, определяющим выбор локального дескриптора из таблицы GDT. Используется при инициализации. LLDT REG/MEM.

SLDT - передает содержимое регистра LDT в операнд-слово. Команда, обратная LLDT. SLDT REG/MEM.

LAR - команда загружает в свой первый операнд байт доступа дескриптора, выбираемый вторым операндом. LAR REG,REG/MEM

LSL - действует аналогично LAR, но загружает в свой первый операнд значение предела выбранного дескриптора.

LSL REG,REG/MEM.

LMSW - загружает из памяти или регистра регистр состояния. Используется для перехода в защищенный режим. Например, LMSW AX.

SMSW - команда, обратная предыдущей.

LTR - загрузка регистра задачи. Выполняется один раз после перехода в защищенный режим. LTR REG/MEM.

STR - сохранение регистра задачи. STR REG,MEM.

CLTS - сброс флага переключения задачи.

LIDT - загрузка регистра таблицы прерываний. LIDT MEM.

SIDT - сохранение регистра таблицы прерываний. SIDT MEM.

VERR - проверка сегмента на чтение. VERR REG/MEM.

VERW - проверка сегмента на запись. VERW REG/MEM.

МП 80386.

Команды для защищенного режима расширились за счет доступа к новым регистрам (CR,TR,DR). Например, MOV CR0,EAX.

МП 80486.

Три описанных ниже команды фактически не являются командами защищенного режима. Точнее, их следует назвать системными.

INVD - очищает внутреннюю кэш-память и выдает специальный цикл шины, который показывает необходимость очистки и внешней кэш-памяти. Данные во внешней кэш-памяти с отложенной записью уничтожаются.

INVLPG - применяется для задания недостоверности одного элемента буфера TLB.

WBINVD - производится очистка внутренней кэш-памяти и формируется специальный цикл шины, показывающий внешней кэш-памяти выполнить обратную запись

ее содержимого в основную память. После этого формируется еще один специальный цикл шины, вызывающий очистку внешней кэш-памяти.

МП PENTIUM.

По-видимому, новых командля защищенного режима не появилось.

IV. 32-битная адресация.

В микропроцессоре 80386 размеры рабочих регистров составляют 32 бита. Соответственно наряду с адресацией типа DS:[BX] стала возможна запись DS:[EBX]. Соответственно размер сегмента уже не ограничивается 64 килобайтами. Ниже представлена программа, демонстрирующая возможность 32-битной адресации.

```
.386
DATA  SEGMENT
TST  DB ?
DATA  ENDS
STAC  SEGMENT  STACK
      DW 50  DUP(?)
STAC  ENDS
CODE  SEGMENT  USE16
      ASSUME  CS:CODE,  DS:DATA,  SS:STAC
BEGIN:
;обнулить  32-битные  регистры
      MOV  EAX, 0
      MOV  EBX, 0
;установить регистр  DS  на  сегмент  DATA
      MOV   AX,DATA
      MOV   DS,AX
;в  ячейке  TST  -  код  буквы  'A'
      MOV   DS:TST,'A'
      PUSH  DS
      LEA   BX,TST
;получить  физический  адрес  сегмента  DATA
      SHL  EAX, 4
;физический адрес  ячейки  TST  в  EBX
      ADD   EBX,EAX
      XOR   AX,AX
;установить регистр  DS  на  начало  памяти
      MOV  DS,AX
;содержимое ячейки  TST  в  DL
      MOV   DL, [EBX]
      MOV   AH,2
      INT   21H
      POP   DS
```

```
;делаем то же самое, но стандартным (сегментным)
;способом
        LEA    BX,TST
        MOV    DL, [BX]
        INT    21H
        MOV AH,4CH
        INT  21H
CODE  ENDS
      . END  BEGIN
```

Рис. 20.1. Пример 32-битной адресации в реальном режиме.

Примеры 32-битного программирования Вы найдете также в главе 25.

V. Страничная адресация.

Сегментная адресация, исповедуемая семейством микропроцессоров INTEL, основана на следующих положениях:

1. Адрес ячейки памяти состоит из двух компонент: адрес сегмента и адрес в сегменте (смещение).
2. Адрес сегмента содержится либо в сегментном регистре (реальный режим), либо в дескрипторе (защищенный режим), на который указывает содержимое соответствующего сегментного регистра.
3. Сегмент и смещение однозначно определяют некоторую величину, называемую линейным адресом. Линейный адрес совпадает с физическим адресом. Физический же адрес ячейки — это просто ее порядковый номер.

В микропроцессоре 80386 появилась так называемая страничная адресация. Страничная адресация работает только в защищенном режиме. Если страничная адресация включена (см. ниже), то линейный адрес, получившийся в результате преобразования сегментного адреса и смещения, подвергается дополнительному преобразованию. В результате линейный адрес может не совпадать с физическим адресом.

В страничном преобразовании базовым объектом является страница длиной 4 Кб. Поскольку линейное адресное пространство 386-го микропроцессора составляет 4 гигабайта, размер реально существующей памяти при этом, естественно, много меньше. Это адресное пространство разделяется на 1 М страниц. Причем реально существует только часть страниц. Несуществующие страницы, которые мы назовем виртуальными, могут храниться на диске и загружаться по мере необходимости. Поскольку размер всех страниц одинаков, это значительно упрощает загрузку и выгрузку страниц. Эту работу, разумеется, выполняет операционная система. Прикладные программы при этом остаются в неведении. В сегментной организации имеется 4 уровня привилегий (см. Главу 5) - О, 1, 2, 3. В страничной организации есть только два уровня: уровень пользователя соответствует уровню 3, и уровень супервизора - соответствует уровням 0, 1, 2 сегментной организации.

В процессе страничного преобразования старшие 20 бит линейного 32-битного адреса заменяются другим значением - номером физической страницы. Младшие 12 бит остаются неизменными. Такое преобразование происходит в два этапа:

1. Старшие 10 бит линейного адреса (плюс 2 нуля справа) указывают на элемент (индексируют) в структуре, которая называется каталогом страниц. Каталог страниц образуется для каждой задачи независимо. Его 20-битный физический адрес содержится в регистре CR3 (биты 12-31, остальные биты считаются нулевыми). Каталог страниц состоит из 32-битных элементов. Каждый элемент указывает на одну из 1024 таблиц страниц.
2. Средние биты 12-21 линейного адреса, дополненные двумя младшими нулями, индексируют, выбирая из нее один из элементов, который указывает уже на конкретную страницу. Каждая из таблиц страниц содержит 32-битные элементы (1024 элементов).
3. В результате "сложения" индекса в каталоге страниц, индекса в таблице страниц и 12-битного смещения получается реальный (физический) адрес.

Элемент каталога страниц и таблицы страниц имеет идентичную структуру и представлен ниже.

31 12									
Адрес таблицы или страницы	9-11	7-8	6	5	4	3	2	1	0

Описание бит:

0 - 1 - страница находится в памяти, 0 - на диске при обращении к странице, которой в памяти нет, генерируется специальное прерывание.

1 - 1 - разрешено чтение-запись, 0 - только чтение.

2 - 0 - уровень супервизора, 1 - уровень пользователя.

3-4 - биты запрещения кэширования страниц и сквозной записи.

5 - 1 - идет обращение к странице для чтения или записи.

6 - 1 - идет обращение к таблице.

7-8 - равны 0.

9-12 - могут использоваться для нужд программистов.

Для уменьшения объема памяти, используемой таблицами страниц, операционная система учитывает реальный объем памяти и строит таблицы страниц, исходя из этого объема, а не из возможного адресного пространства. Причем уменьшается количество таблиц страниц, а не их размер.

Ниже приводится последовательность команд, разрешающих страничную адресацию:

```
      PUSHFD
      CLI
      MOV  EAX,KAT_ADR  ;физический адрес каталога
      MOV  CR3,EAX
      MOV  EAX,CRO      ;разрешить страничную адресацию
      BTC  EAX,31
      MOV  CR0,EAX
      JMP  NEXT
NEXT:
      POPFD
```

Добавим также, что для микропроцессоров Pentium появилась возможность оперировать со страницами размером в 4 Мб.

На этом мы заканчиваем рассмотрение семейства процессоров INTEL. Остаток главы посвящен рассмотрению программирования с использованием арифметического сопроцессора.

VI. Арифметический сопроцессор.

Один ум хорошо, а два - лучше.

Русская пословица.

Представленный ниже материал вполне достаточен для программирования арифметического сопроцессора. Для заинтересованных читателей укажу также книги, где можно получить исчерпывающие сведения: [2,17,32]. Надо иметь в виду, что, начиная с процессора 80486, сопроцессор стал встроенным. В этой связи значение его резко возросло. На современном компьютере Вы можете смело использовать его команды. В конце приложения дается полный список таких команд, а также другая справочная информация о сопроцессоре.

Арифметический сопроцессор является вспомогательным процессором и не может работать самостоятельно, и не может осуществлять самостоятельно выборку команд. Команды его могут идти вперемежку с командами микропроцессора. Оба процессора подключены к системной шине. Выборку команд осуществляет микропроцессор, и все команды попадают в оба процессора, но каждый выполняет свои. Микропроцессор, однако, принимает некоторое участие в выполнении команд сопроцессора. Если в команде сопроцессора требуется вычислить физический адрес или необходимо считать данные из памяти, то микропроцессор делает это, выставляя адрес или данные на шину, откуда их считывает сопроцессор. То есть связь между процессорами осуществляется через шину.

Из сказанного становится ясно, что сопроцессор в силу своей несамостоятельности синхронизирует свои действия с действиями микропроцессора, т.е. ждет, когда микропроцессор сделает выборку очередной команды. Однако необходима и обратная синхронизация. Необходимость эта возникает в двух случаях:

а) синхронизация по командам - передавать очередную команду сопроцессору можно, только если он готов к выполнению, т.е. выполнил последнюю команду;

б) синхронизация по данным - микропроцессор использует данные, которые используются также сопроцессором, данные должны быть готовы. Такая связь осуществляется по линии TEST. Активный сигнал на линии говорит о готовности сопроцессора. Проверку этого сигнала осуществляет команда WAIT (см. главу 4).

Синхронизацию по командам помогает осуществлять ассемблер. Когда Вы, например, в своей программе пишите команду FLDZ (см. ниже), то ассемблер автоматически генерирует следующую последовательность: WAIT/FLDZ. Команды сопроцессора можно вводить и с помощью мнемоники ESC. Например, можно записать команду FIDIV [SI] - разделить число, находящееся в вершине стека (см. ниже) на целое из ячейки памяти [SI].

После трансляции ассемблер, как уже было сказано сгенерирует две команды: WA1T/FIDIV [SI]. Если же мы запишем в программе директиву ESC 16H,[SI], то будет сгенерирована всего одна команда FIDIV.

Программная модель арифметического сопроцессора имеет стековую основу - восемь 80-битных регистра, взаимодействие между которыми носит стековый характер. Трехбитовый указатель стека ST (или ST(0)) направлен на регистр, являющийся в данный момент вершиной. При операции включения в стек осуществляется декремент указателя, а в новую вершину стека помещаются загружаемые данные. При операции извлечения из стека в получатель из вершины извлекается данное, а затем осуществляется инкремент указателя. Стек имеет кольцевую структуру. Поэтому если указатель был равен 000B и в стек положили данное, значение указателя станет 111B. В командах сопроцессора допускается явное или неявное обращение к вершинам стека. Так команды с одним операндом (унарные) предполагают некоторую операцию (вычисление корня, синуса и т.д.) с содержимым регистра, находящегося в вершине. Результат команды при этом помещается в этот же регистр. В бинарных же операциях (два операнда) часто операндами являются два верхних элемента стека, а результат помещается на место одного из них. Возможна и прямая адресация к элементам стека: FADD ST,ST(2) - складывает содержимое верхнего регистра с регистром, отстоящим от него на 2 (например, R3 с R5), и результат помещается в вершину.[56]

С каждым регистром сопроцессора связано двухбитовое поле, называемое тэгом. Тэг фиксирует наличие в регистре действительного ненулевого числа - 00B, истинного нуля - 01B, специального числа - 10B, отсутствие данных - 11B. Тэг используется сопроцессором, в частности, для фиксирования особых случаев (загрузка несуществующего числа, запись в занятый регистр и т.д.).

Ниже представлена простая программа суммирования элементов целочисленного массива. Результат заносится в ячейку SUM. Для проверки результата мы выводим символ, соответствующий этому числу. Т.к. результат должен быть равен 65, то и напечатана должна быть буква «А». Как уже отмечалось, перед командами сопроцессора автоматически ставится команда WAIT. Нам, однако, приходится использовать эту команду и отдельно - перед выводом результата. Делается это для того, чтобы сопроцессор успел передать данные в ячейку памяти. Данная программа является иллюстрацией работы сопроцессора, а для выполнения действий с целыми числами более эффективен обычный микропроцессор. Сопроцессор эффективен при работе с числами в вещественном формате.

```
CODE  SEGMENT
      ASSUME CS:CODE,DS:CODE
      ORG 100H
BEGIN:
      JMP SHORT BEG
;массив для сложения
ARRAY DW 25,1,1,1,1,1,5,5,5,3,5,7,5
```

---

[56] **В основу вычисления арифметического сопроцессора положен так называемый принцип «обратной польской записи».**

```
;сюда помещаем сумму
SUM    DW ?
BEG:
       MOV CX,13
;инициализация сопроцессора
;предполагает, в частности, очистку всех регистров (тэг=11B)
       FINIT
;поместить в вершину стека 0
;здесь будет накапливаться сумма
       FLDZ
;с помощью регистра SI будем адресовать массив
       XOR SI,SI
MORE:
;осуществить сложение, а результат поместить в вершину стека
       FIADD DWORD PTR ARRAY[SI]
       ADD  SI,2
       LOOP MORE
;поместить результат в ячейку SUM
       FIST DWORD PTR SUM
/ждать сопроцессор
       WAIT
/вывести результат суммирования, как символ
       MOV  AH,2
       MOV  DL,BYTE PTR SUM
       INT  21H
       RET
CODE ENDS
       END BEGIN
```

Рис. 20.2. Пример использования сопроцессора.

Следующий пример написан на Си++. Это объясняется тем, что формат вещественных чисел довольно сложен. Пришлось бы писать процедуру преобразования из формата сопроцессора в ASCII-формат, что значительно бы усложнило программу. С другой стороны, формат вещественных чисел сопроцессора практически совпадает с форматом вещественных чисел, которым оперирует Си. Кроме того, там есть хорошая функция PRINTF, позволяющая выводить на экран числа в различных форматах[57].

[57] Не попадитесь на удочку компилятора Си. Если трансляция проводится в режиме эмуляции сопроцессора, то команды сопроцессора будут заменены командами микропроцессора 8088/8086, а программа будет работать, как будто сопроцессор есть. Поэтому после отладки проверьте программу отладчиком.

```
#include <stdio.h>

void main(void)
{
int num=200;
double sq;
/*далее идет ассемблерный код*/
asm {
/* инициализация */
/*предполагает, в частности, очистку всех регистров
 (тэг=11B)*/
      FINIT
/*поместить в вершину стека 0*/
/*здесь будет находиться число, а потом его корень*/
       FLDZ
/*загрузить в вершину стека число*/
      FILD num
/*извлечь корень*/
      FSQRT
/*передать в ячейку памяти*/
      FST sq
/*подождать сопроцессор*/
       WAIT
  }
/*вывести результат*/
printf("%f",sq);
};
```

Рис. 20.3. Пример использования сопроцессора.

Программа извлекает квадратный корень из целого числа 200. Практически вся программа написана на ассемблере, а из языка Си используется лишь PRINTF. Программу можно компилировать в любой модели, кроме TINY. Рассмотрим некоторые детали программы. Команда FILD NUM - загрузить из переменной NUM целое число в вершину стека сопроцессора. Указатель стека при этом уменьшается на 1, что в данной задаче несущественно. Команда FST SQ - передать содержимое вершины стека в вещественную переменную SQ. Указатель стека при этом не меняется. Команда FSQRT - извлечь корень из содержимого вершины стека и поместить туда же. Указатель стека не меняется. Команда WAIT перед PRINTF необходима, чтобы дождаться, когда ячейка SQ получит результат.

При использовании сопроцессора имейте в виду, что неумелое применение его команд может привести не к ускорению, а к замедлению работы программы. Такие действия, как сложение и вычитание над целыми числами, следует делать средствами обычного микропроцессора. Сопроцессор же нужно привлекать для операций деле-

ния различных функций. Некоторое время назад, когда значительное количество компьютеров шло без сопроцессора, использование его в программах было вообще нежелательно.

Ниже приводится программа проверки сопроцессора на вычисление экспоненциальных функций. Программу следует откомпилировать в режиме использования сопроцессора. Чтобы облегчить понимание программы укажу, что у арифметического сопроцессора есть функция вычисления 2^X - 1. Для того чтобы получить EXP(X), нужно 2 возвести в степень, равную X, умноженному на логарифм по основанию 2 от константы E. Этот логарифм может загружаться в регистры сопроцессора специальной командой (см. программу).

```
#include <stdio.h>
#include <math.h>

double fun(double i)  /*вычисление экспоненты, значение
0<=I<=0.5*/
{
double ex;
asm {
FINIT               /*инициализация*/
FLD i               /*загрузка числа*/
FLDL2E              /*загрузка константы*/
FMUL ST,ST(1)       /*находим произведение*/
F2XM1               /*вычисляем EXP(X)-1*/
FST  ex             /*передать в ячейку*/
WAIT
};
ex+=1;              /*получаем EXP*/
return(ex);
};

void main(void)
{
/*проверка работы сопроцессора*/
printf("%f\n",fun(0.35));
printf("%f\n",fun(0.4));
printf("%f\n",exp(0.35));
printf("%f\n",exp(0.4));
};
```

Рис. 20.4. Справочные данные по арифметическому сопроцессору. Справочник команд арифметического сопроцессора 8087.

Арифметические команды.

Команда	Комментарий	Действия
FADD dst,src	Сложить вещественные числа	dst<-dst)+(src)
FADDP dst,src	Сложить вещественные числа и извлечь из стека	dst<-dst)+(src) ST <- (ST)+1
FIADDsrc	Прибавить целое чис.	ST(0)<-ST(0)+(src)
FSUBdst,src	Вычесть вещественные числа	dst<-(dst)-(src)
FSUBP dst,src	Вычесть вещественные числа и извлечь из стека	dst<-(dst)-(src) ST <- (ST)+1
FSUBR dst,src	Обратное вычитание	dst<-(src)-(dst)
FSUBRPdst,src	Обратное вычитание с извлечением из стека	dst<-(src)-(dst)ST<-(ST)+1
FISUBsrc	Вычесть целое число	ST(0)<-ST(0)-(src)
FISUBRsrc	Обратное вычитание	ST(0)<-(src)-ST(0)
FMULdst,src	Умножить вещественные числа	dst<-(dst)*(src)
FMULP dst,src	Умножить с извлечением из стека	dst<-(dst)*(src)ST<-(ST)+1
FIMUL src	Умножить на целое число	ST(0)<-ST(0)*(src)
FDIVdst,src	Разделить вещественные числа	dst<-(dst)/(src)
FDIVPdst,src	Разделить с извлечением из стека	dst<-(dst)/(src) ST <- (ST)+1
FDIVRdst,src	Обратное деление	dst<-(src)/(dst)
FDIVRPdst,src	Обратное деление с извлечением из стека	dst<-(src)/(dst) ST <- (ST)+1
FIDIVsrc	Разделить на целое число	ST(0)<-ST(0)/src
FIDIVR src	Обратное деление	ST(0)<- src/ST(0)
FABS	Абсолютное значение	ST(0)<-|ST(0)|
FCHS	Сменить знак	ST(0)<--ST(0)
FPREM	Частичный остаток	ST(0)<-ST(0)MODST(1)
FRNDINT	Целая часть	целая ч. ST(0)
FSCALE	Масштабировать	ST(0)<-ST(0)^ST(1)
FSQRT	Извлечь корень	ST(0)<-sqrt(ST(0))
FXTPACT	Выделить порядок и мантиссу	ST(0)<-поряд.ST(0) ST(1)<-мант.ST(0)

Команды передачи данных

Команда	Комментарий	Действия
FLDsrc	Загрузить вещественное число	ST <- (ST)-1 ST(0) <- (src)
FILDsrc	Загрузить целое число	ST <- (ST)-1 ST(0) <- (src)
FBLD src	Загрузить десятичное число	ST <- (ST)-1 ST(0) <- (src)
FST dst	Запомнить вещественное число	dst <- ST(0)
FIST dst	Запомнить целое число	dst <- ST(0)
FBSTP dst	Запомнить десятичное число и извлечь из стека	dst <- ST(0) ST <- (ST)+1
FSTP dst	Запомнить вещественное число и извлечь из стека	dst <- ST(0) ST <- (ST)+1
FISTP dst	Запомнить целое число и извлечь из стека	dst <- ST(0) ST <- (ST)+1
FXCH dst	Обмен	ST(0)<->(dst)

Команды сравнения.

Команда	Комментарий	Действия
FCOM src	Сравнить вещественные числа	ST(0) - src Установить C3, CO
FCOMP src	Сравнить и извлечь из стека	ST(0) - src ST<-ST+1 Установить C3, CO
FCOMPP	Сравнить и дважды извлечь	ST(0) - ST(1) ST <- ST+1 Установить C3, CO
FICOM src	Сравнить с целым	ST(0) - src Установить C3, CO
FICOMP src	Сравнить с извлечением	ST(0) – src ST <- ST+1 Установить C3, CO
FTST	Проверить вершину стека	ST(0) - 0.0 Установить C3, CO
FXAM	Проанализировать вершину стека	Установить C3, C2, C1, CO

Трансцендентные функции.

Команда	Комментарий	Действия
F2XM1	Вычислить 2^x – 1	ST(0)<-2^ST(0)-1
FPATAN	Вычислить частичный арктангенс	Z<-arctg(ST(0)/ST(1)) ST<-ST+1 ST(0)<-Z
FPTAN	Вычислить частичный тангенс	tgST(0)=Y/X ST(0)<-Y ST<-ST-1
FYL2X	Вычислить Y*lg	ST(0)<-X ST<-ST+1 ST(0)<-Z
FYL2XP1	Вычислить Y*ln	ST<-ST+1 ST(0)<-Z

Команды загрузки.

Команда	Комментарий	Действия
FLDZ	Загрузить нуль	ST<-ST-1ST(0)<-0.0
FLD1	Загрузить 1	ST<-ST-1 ST(0)<-1.0
FLDPI	Загрузить Пи	ST<-ST-1 ST(0)<-3.14....
FLDL2E	Загрузить log	ST(0)<-log
FLDL2T	Загрузить log	ST(0)<-log
FLDLG2	Загрузить log	ST(0)<-log
FLDLN2	Загрузить log	ST(0)<-log

Команды управления сопроцессором.

Команда	Комментарий	Действия
FINIT/FNINIT	Инициализировать сопроцессор	Слово сос.<-03FF Слово тэгов<-NULL Все флажки<-0 ST<-0
FDISI/FNDISI	Запретить прерывания	IEM<-1
FENI/FNENI	Разрешить прерывания	IEM<-0
FCLEX/FNCLEX	Сбросить флажки особых случаев	флажки<-0
FINCSTP	Инкремент указателя	ST<-(ST)+1
FDECSTP	Декремент указателя	ST<-(ST)-1
FSTSW/FNSTSW dst	Запомнить слово состояния	dst<-слово сос.
FSTCW/FNSTCW dst	Запомнить слово управления	dst<-слово упр.
FLDCW src	Загрузить слово управления	слово упр.<-src
FSTEN/FNSTEN dst	Запомнить среду	dst<-слово упр. dst+2<-слово сос. dst+4<-слово тэгов dst+6<-указ.команд dst+ 10<-указ.операн.
FLDENV src	Загрузить среду	слово упр.<-src слово сос.<-src+2 слово тэгов<-src+4 указ.команд<-src+6 указ.операн.<-src+10
FFREE dst	Освободить регистр	ST(i) - пустой
FNOP	Холостая команда	

FSAVE/FNSAVE dst	Запомнить полное состояние	dst<-слово упр. dst+2<-слово сос. dst+4<-слово тэгов dst+6<- указ.команд dst+10<-указ.операн. dst+14..dst+84<-ST(0), ST(1),ST(2)... Инициализировать сопроцессор
FRSTORE src	Восстановить полное состояние	слово упр.<-src слово сос.<-src+2 слово тэгов<-src+4 указ.команд<-src+6 указ.операн.<-src+10 ST(0),ST(1)..<-<-src+14..src+84

Программная модель арифметического сопроцессора.

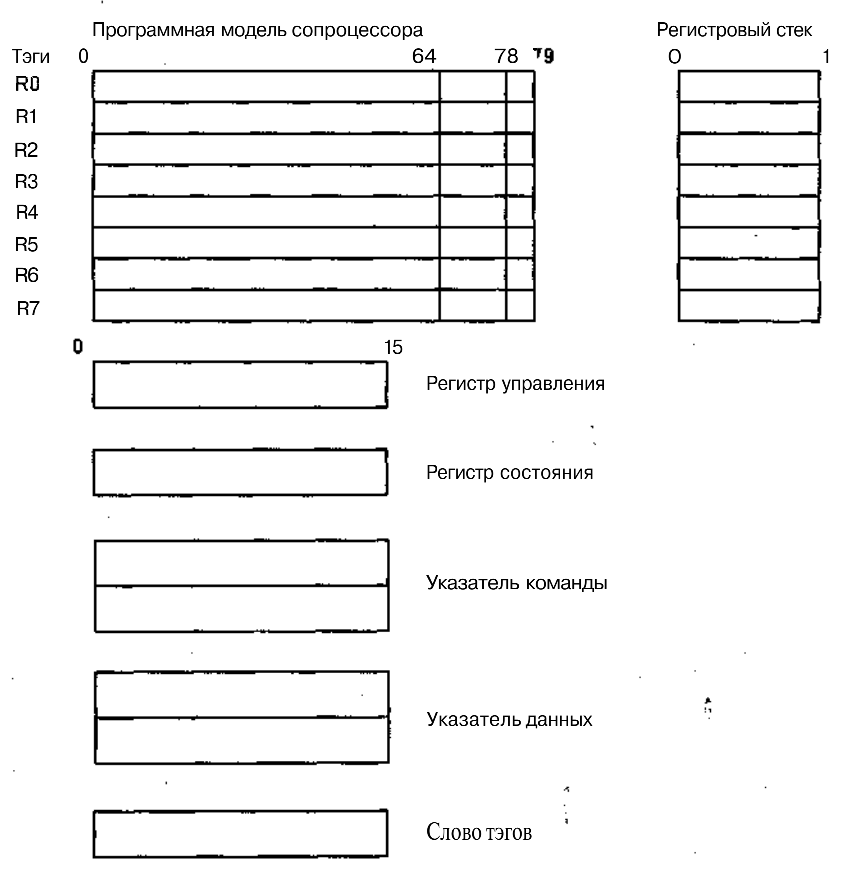


Рис. 20.5. Программная модель арифметического сопроцессора.

Регистровый стек. Восемь 80-битных регистров, где числа хранятся во временном вещественном формате. Трехбитовое поле ST из регистра состояния определяет регистр, являющийся вершиной стека. Стек имеет кольцевую структуру.

Двухбитовый тэг. Ассоциируется с каждым регистром стека. Совокупность тэгов образует слово тэгов. Значение тэгов:

00 - конечное не нулевое действительное число,

01 - истинный нуль,

10 - не число или бесконечность,

11 - регистр пуст.

Программист может интерпретировать содержимое регистров стека путем анализа слова тэгов.

Слово управления.

Первые 6 бит составляют маску особых случаев. Если один из этих бит равен 1, то при возникновении соответствующего особого случая не возникает прерывания.

бит 0 - недействительная операция,

бит 1 - случай денормализованного операнда,

бит 2 - деление ненулевого значения на О,

бит 3 - переполнение,

бит 4 - антипереполнение,

бит 5 - потеря точности.

Остальные биты:

бит 6 - резерв,

бит 7 - 1 - запретить прерывание центрального процессора,

биты 8-9:

00 - точность 24 бита,

10 - 53 бита,

11 - 64 бита.

биты 10-11:

00 - округление к ближайшему,

01 - округление вниз,

10 - округление вверх,

11 - отбрасывание.

Бит 12 - интерпретация бесконечности:

0 - беззнаковая бесконечность,

1 - бесконечность со знаком.

Слово состояния. Является в некотором роде отражением регистра управления. Кроме того, здесь есть биты условия: СО - 8-й бит, С1 - 9-й бит, С2 - 10-й бит, С3 - 14-й бит. Вам не нужно анализировать условие. Биты условий вначале передаются в память, затем в регистр флажков микропроцессора. Далее используется обычная команда условного перехода.

Дальнейшее развитие сопроцессоров.

Здесь приводится краткий обзор развития сопроцессоров, начиная с 287-го и заканчивая сопроцессором Pentium.

Сопроцессор 80287.

Основные отличия:

1. Может работать в реальном и защищенном режиме.
2. Команда WAIT, встроенная во все команды сопроцессора, поэтому применять ее отдельно не стоит.
3. Имеются отличия в слове состояния. Бит 6 устанавливается при некорректной работе со стеком. Бит 7 устанавливается при возникновении немаскированного исключения.
4. Из управляющего слова удалены биты контроля над бесконечностью, а также бит 7.
5. Новые команды в сопроцессоре 287 отсутствуют.

Дальнейшее развитие сопроцессоров.

Здесь приводятся те изменения, которые произошли с сопроцессором в дальнейшем, не конкретизируя, к какому сопроцессору это относится (387-му, 487-му или ...).

1. Новые команды: FUCOMP st(i) - сравнение st(0) с st(0) без учета порядков и выталкивание из стека; FUCOMPP st(i) - сравнение st(0) с st(0) без учета порядков и двойное выталкивание из стека; FPREM1 - нахождение частичного остатка в стандарте IEEE; FCOS - косинус st(0); FSIN - синус st(0); FSINCOS - синус и косинус (st(1)<-sin(st(0), st(0)<-cos(st(0)).

Глава 21. Программирование в локальных сетях.

„тятя, тятя, наши сети...
А.С.Пушкин

Данную главу следует рассматривать как введение в программирование сетевых протоколов IPX и SPX, а также управление файловой системой в сети. В целом же материал по локальным сетям столь обширен, что для изложения его не хватит и всей нашей книги. Поэтому, если Вы всерьез собираетесь заниматься программированием в локальных сетях, рассматривайте эту главу как промежуточный этап. В локальных сетях Microsoft Windows следует использовать сетевые API функции, которые можно найти в соответствующем справочнике (см. главы 24,25). Данная глава рассматривает программирование на рабочих станциях, подключенных к сети Novel и работающих под управлением операционной системы MS DOS.

I. Общие замечания.

Все рассмотрения в этой главе касаются в первую очередь сетевой операционной системы NOVEL NETWARE. Однако практически весь материал, относящийся к работе с протоколами IPX и SPX, носит более универсальный характер. В частности, его можно использовать и с такими системами, как NWLite и Personal Netware.

Конечно, если Ваша программа предполагает использование возможностей локальной сети, то она в первую очередь должна определить, подключен компьютер к сети или нет. Наиболее просто это можно сделать, используя функцию 44H DOS. Подфункция 9H этой функции как раз и определяет, является данное устройство сетевым или нет. Ниже (Рис. 21.1) представлена простая программа, определяющая наличие в системе сетевых устройств. Данная программа будет работать не только с NOVEL'овскими сетями.

```
CODE  SEGMENT
       ASSUME CS:CODE, DS:CODE, SS:CODE
       ORG  100H
BEGIN:
       MOV  BL,1
       MOV  CX,33
LOO:
       MOV  AX,4409H
       INT  21H
       AND  DX,1000H
       JNZ  YES
       INC  BL
       LOOP  LOO
```

```
;сетевых устройств не обнаружено
      LEA  DX,TEXT1
      MOV  AH,9
      INT  21H
      RET
YES:
      LEA  DX,TEXT2
      MOV  AH, 9
      INT  21H
      RET
;блок сообщений
TEXT1  DB  'Сетевых устройств не обнаружено.', 13,10, '$'
TEXT2  DB  'Найдены сетевые устройства.', 13,10,'$'
CODE  ENDS
      END  BEGIN
```

Рис. 21.1. Простая программа, позволяющая определить наличие сетевых устройств.

Имейте, однако, в виду, что драйвер устройства CD-ROM откликается так же, как сетевое устройство. И здесь можно только посоветовать обратиться к функции 15H этого драйвера (прерывание 2FH, подфункция 0). В BX эта функция возвращает количество устройств CD-ROM. Если в BX 0, то здесь все ясно. В противном случае Вам придется проводить дополнительные исследования, используя функции драйвера CD-ROM. Впрочем, познакомившись с материалом, представленным ниже, Вы, без сомнения, найдете сетевые функции, которые помогут определить, является данное устройство сетевым или нет, например, функции, обслуживающие файловую систему.

Наличие сетевого устройства не является необходимым условием того, что данный компьютер является сетевой рабочей станцией. Более полную и точную информацию о сети можно получить, используя диагностические средства IPX протокола (см. ниже).

О сети NOVEL.

Не вдаваясь в подробное описание локальных сетей (ЛС), рассмотрим основные особенности сети на базе ОС NOVEL NETWARE. Технические особенности - платы, кабеля, репитеры и т.п. здесь упоминаться не будут вообще. Предполагается, что читатель хорошо подготовлен для работы в сети как с теоретической, так и с практической позиции. Итак, вот эти особенности:

1.Данная сеть является сетью с централизованным управлением. Это означает, что среди всех компьютеров сети выделены один или более. На них работает специальная многозадачная операционная система. Они управляют работой ЛС, а также предоставляют ресурсы компьютера, на котором она работает, другим компьютерам. Такие выделенные компьютеры называются серверами. В одноранговых сетях (Personal Netware, Lantastic, локальная сеть Windows и др.) любая из рабочих станций может

одновременно быть и сервером. Поэтому почти все, что далее будет говориться о работе с сервером, будет в значительной степени относиться и к серверам в одноранговой сети.

2. Другие компьютеры называются рабочими станциями. На них работает обычная операционная система MS DOS или какая-нибудь другая ОС. Кроме того, чтобы компьютер стал рабочей станцией, на нем должны быть запущены специальные программы, которые и осуществляют его вхождение в сеть. Эти программы, постоянно находясь в памяти, позволяют осуществлять управление локальной сетью с рабочей станции.

3. После подсоединения к сети рабочая станция получает возможность использовать дисковые ресурсы серверов. Это осуществляется посредством появления на компьютере одного или нескольких сетевых устройств, которые работают в основном так же, как и другие блоковые устройства - гибкие диски и разделы жестких дисков. Посредством выделения общих сетевых устройств рабочие станции могут взаимодействовать друг с другом - использовать общие БД и т.п. Кроме того, рабочие станции могут взаимодействовать друг с другом посредством специальных сетевых протоколов (см. ниже).

Можно выделить пять уровней программирования в локальной сети:

1. Программа не знает, что она работает в локальной сети. Поскольку сетевые диски по своим свойствам в основном не отличаются от обычных разделов жесткого диска, то для многих целей при программировании не стоит задумываться над тем, будет программа использоваться в сети или нет.

2. Программа знает, что она работает в локальной сети. Например, для ее работы требуется знать, является устройство сетевым или нет.

3. Программы, работающие на разных рабочих станциях, должны синхронизировать свою работу. Допустим, они работают с общей базой данных. Здесь может возникнуть конфликт: например, при одновременном редактировании одной и той же записи. Программы могут блокировать и разблокировать записи и файлы и использовать механизм транзакций.

4. Программы, работающие на разных рабочих станциях, обмениваются данными, минуя сервер, используя один из сетевых протоколов.

5. Программы могут использовать ресурсы сервера: подключаться к сети, менять пароли, иметь дополнительные возможности управление файловой системой.

Протоколы передачи.

Связь между программами на рабочих станциях может осуществляться по одному из следующих протоколов: IPX, SPX.[58] Существуют и другие протоколы, но мы с ними работать не будем. Поддержку работы того или иного протокола осуществляют со стороны рабочей станции резидентные драйверы. Ниже представлена программа, определяющая присутствие поддержки протоколов IPX и SPX. Вначале программа определяет точку входа - POINT. Через эту точку входа осуществляется вызов функций поддержки протоколов IPX и SPX. Прерывание же 2FH является лишь средством оп-

[58] Протокол NetBios несколько устарел для современных разветвленных локальных сетей.

ределения этой точки входа. Может возникнуть вопрос: почему и саму связь с драйвером не осуществлять через это прерывание? Ответ прост: это прерывание могут перехватывать многие программы, и от этого скорость его работы может сильно замедлиться.

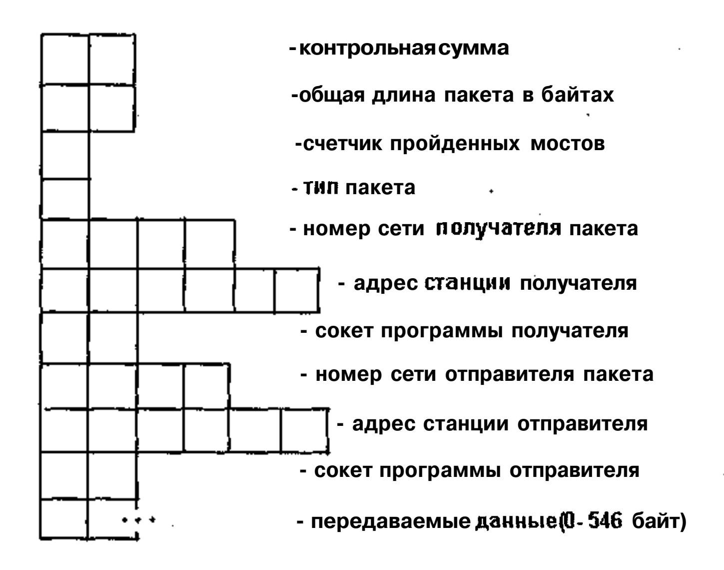
```
CODE SEGMENT
        ASSUME CS:CODE,DS:CODE
        ORG 100H
BEGIN:
;проверить присутствие сетевого драйвера
        MOV AX,7A00H
        INT 2FH
        CMP AL,0FFH
        JZ  OK1
        LEA DX,TEXT2
        JMP SHORT _END
OK1:
        LEA DX,TEXT1
        CALL WRITE
;сохранить точку входа в драйвер
        MOV  POINT,DI
        MOV  POINT+2,ES
        MOV  AX,0
        MOV  BX,10H    ;функция SPX
;проверяем наличие протокола SPX
        CALL DWORD PTR POINT
        CMP  AL,0FFH
        JZ   OK2
        LEA  DX,TEXT4
        JMP  SHORT _END
OK2:
        LEA  DX,TEXT3
_END:
        CALL WRITE
        RET
;процедура вывода строки
WRITE  PROC
        MOV AH,9
        INT 21H
        RET
WRITE ENDP
;сообщения
TEXT1 DB 'Протокол IPX присутствует.', 13,10,'$'
TEXT2 DB 'Протокол IPX отсутствует.', 13,10,'$'
```

```
TEXT3 DB 'Протокол SPX присутствует.',13,10,'$'
TEXT4 DB 'Протокол SPX отсутствует.',13,10,'$'
;точка входа в драйвер
POINT DW ?
      DW ?
CODE ENDS
      END BEGIN
```

Рис. 21.2. Программа, определяющая присутствие драйверов поддержки протоколов IPXи SPX.

П. Протокол IPX. Пакет в протоколе IPX.

Данные в протоколе IPX (как, впрочем, и в других протоколах) передаются в виде пакетов. Ниже показана структура этого пакета.



Контрольная сумма - 2 байта: хранится контрольная сумма передаваемых пакетов, подсчет контрольной суммы осуществляет сетевой драйвер.

Общая длина пакета - 2 байта, длина пакета вместе с заголовком. Длина заголовка составляет 30 байт. Длину определяет сетевой драйвер.

Счетчик пройденных мостов - 1 байт, каждый раз, когда пакет проходит через мост, значение счетчика увеличивается на 1.

Тип пакета -1 байт, протоколу IPX соответствует значение байта 4.

Номер сети получателя - 4 байта, заполняется передающей программой.

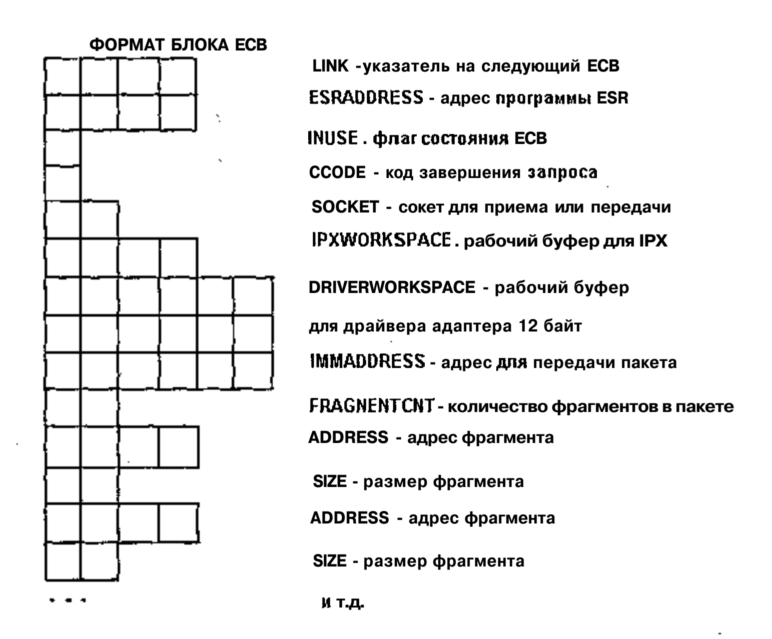
Адрес станции получателя - 6 байт, заполняется передающей программой.

Сокет программы получателя - 2 байта, предназначен для адресации программы, которой предназначается данный пакет. Заполняется передающей программой.

Номер сети отправителя - 4 байта, заполняется передающей программой. Адрес станции отправителя - 6 байт. Сокет программы отправителя - 2 байта. Данные - от 0 до 546 байт.

Адрес программы в сети складывается из трех составляющих: адрес сети (сегмента сети), адрес рабочей станции (определяется сетевой платой), адрес программы, который мы будем называть сокетом (SOCKET)[59]. Необходимость введения сокета обусловлена возможностью работы на рабочей станции одновременно нескольких программ. Поскольку для хранения сокета используется двухбайтовое число, то его диапазон от 0 до FFFFH. Диапазон 0-3000H зарезервирован за конкретным программным обеспечением фирмой XEROX, автором протокола IPX. В свою очередь, фирма NOVEL также резервирует часть сокетов, так что для обычного программного обеспечения остается диапазон больший 4000H и меньший 8000H. Если Вы связываете рабочие станции, находящиеся в разных сегментах (сетях), то Вам понадобится, кроме знания адресов сегментов (сетей), также адрес моста. Адрес моста определяется сетевой платой, которая играет роль моста.

Для приема и передачи пакета сетевая программа должна создать EVENT CONTROL BLOCK (ECB) - блок управления событием. Подготовив такой блок, программа передает его адрес сетевому драйверу. Ниже представлен формат этого блока.



[59] Термин "сокет" довольно непривычен нашему уху. Встречается в литературе и другое название - "гнездо", что и является переводом английского слова "сокет".

Рассмотрим ряд замечаний по содержимому ECB.

LINK - ссылка на следующий блок. Используется сетевым драйвером.

ESRADDRES - адрес подпрограммы, которая будет вызвана по завершению процесса передачи или получения пакета.

ESR - EVENT SERVICE ROUTINE.

INUSE - при завершении операции флаг получает нулевое значение. Проверяя флаг, можно определить, завершена операция передачи или приема пакета. Вот возможные значения этого поля:

FF - ECB используется для передачи пакета,

FE - ECB используется для приема пакета,

FD - ECB используется функциями асинхронного управления событиями,

FB - пакет данных передан или принят, но ECB находится во внутренней очереди IPX.

CCODE - код результата,

при приеме данных:

00 - пакет принят без ошибок,

FF - сокет не был предварительно открыт,

FD - переполнение пакета,

FC - запрос на прием данных был отменен специальной функцией;

при передаче данных:

00 - данные переданы без ошибок,

FF - пакет невозможно передать физически,

FE - пакет невозможно доставить по назначению,

FD - произошел сбой,

FC - запрос на передачу данных был отменен специальной функцией.

SOCKET - сокет данной программы,

IPXWORKSPACE - зарезервировано для сетевого драйвера,

DRIVERWORKSPACE - зарезервировано для сетевого драйвера,

IMMADDRES - адрес рабочей станции назначения либо адрес моста, если адресуемая рабочая станция находится в другой сети (через мост).

FRAGMENTCNT - количество фрагментов, на который разбивается передающий пакет, обычно 2 (см. ниже).

Хочу заметить, что все адреса и в пакете, и в ECB должны указываться в обратном порядке (см. ниже).

III. Сетевые функции IPX.

Ниже представлены сетевые функции, позволяющие программам осуществлять связь через протокол IPX.

Открыть сокет.

Вход:

BX=0 AL=0 - коротко живущий (закрывается по окончанию работы программы),

0FFH - долго живущий.

Выход:

AL=0 - сокет открыт,

FFH - сокет был открыт раньше,
FEH - переполнилась таблица сокетов.
DX - назначенный номер гнезда.

Закрыть сокет.
Вход:
BX=1,
DX - номер закрываемого сокета.

Любые события, связанные с этим сокетом отменяются. Закрытие уже закрытого сокета не вызывает никаких последствий.

Вычисление непосредственного адреса.
BX=2

ES:SI - указатель на буфер длиной 12 байт, содержащий полный сетевой адрес станции, на которую будет послан пакет.

ES:DI - указатель на буфер длиной 6 байт, в который будет записан непосредственный адрес (либо рабочей станции, либо моста).

Выход:
CX - время доставки,
AL - 0 - успешно,
FEH - нет пути к адресу назначения.

Вычисление своего адреса.
BX=9

ES:DI - указатель на буфер длиной 10 байт, куда будет записан адрес станции, на которой работает данная программа (без сокета).

Структура:
DEST_NUMBER DB 4 DUP(?)
DEST_NODE DB 6 DUP(?)

Принять пакет.
BX=4
ES:DI - указатель на заполненный блок ECB. Необходимо заполнить поля:
ESRADDRES;
SOCKET;
FRAGMENTCNT;
указатель на буферы фрагментов,
размеры фрагментов.
Код завершения:
AL - 0 - успешно,
FFH - сокет не найден.

Послать пакет.
BX=3
ES:DI - указатель на заполненный блок ECB. Необходимо заполнить поля:
ESRADDRES;
SOCKET;
IMMADDRES;

FRAGMENTCNT;
указатели на буферы фрагментов ADDRES;
размеры фрагментов.
В заголовке пакета необходимо заполнить:
PAKETTYPE;
DESTNETWORK;
DESTNODE;
DESTSOCKET.

Освобождение канала связи.
BX=0BH
ES:SI - указатель на структуру, содержащую сетевой адрес станции.

Функция посылает сообщение сетевому драйверу, что данная программа больше не будет посылать пакеты на указанную станцию. Не вызывается из ESR.

Функция сброса поля INUSE.
BX=5,
AX - время задержки в тиках,
ES:SI - указатель на блок ECB.

Функция немедленно возвращает управление вызвавшей программе, а через указанный промежуток времени сбрасывается **в нуль флаг** INUSE, и вызывается программа ESR (если таковая предполагалась).

Функция измерения временных интервалов.
Вход:
BX=8
Выход:
AX - интервальный маркер.

Для измерения времени между двумя событиями. Функция возвращает засечку времени, измеряемую в тиках таймера (в 1 с. прим. 18.2тика). Следующее обращение к этой функции вернет засечку с большим значением. Вычитая одно значение из другого, можно получить значение интервала времени. Этой командой, естественно, не могут измеряться интервалы большие одного часа.

Функция отмены ожидания.
Вход:
BX=6,
ES:SI - указатель на блок ECB.
Выход:
AL=00 - без ошибок,
F9H - обработка ECB не может быть отменена,
FFH - указанный ECB не используется.

Отменяет ожидание события с указанным ECB. ESR не вызывается. CCODE устанавливается в FCH.

Функция выделения сетевому драйверу необходимого времени.
BX=0AH
Вызывается в цикле ожидания (см. ниже в программах).

Ниже представлены две сетевые программы. Программа 1 - ожидает прихода сообщения от программы 2. Предполагается, что программа 2 знает полный адрес программы 1. В частности, известен адрес рабочей станции, на которой запущена программа 1. Обратите внимание на то, как записаны в программе все адреса. Передаваемый пакет разбивается на два блока FRAGNENTCNT=2. В первом блоке передается заголовок пакета, имеющий всегда фиксированную длину, равную 30 байтам. Во втором блоке передается строка. Длина второго блока составляет 32 байта (SIZ=32). Максимальная длина блока данных составляет - 540 байт. Ожидающая программа в цикле проверяет поле INUSE - нулевое значение является признаком, что пакет получен. Обращаю Ваше внимание на то, что отправляющая программа также проверяет поле INUSE. При успешной посылке пакета поле также обращается в 0. Речь идет именно о посылке пакета. Принят ли пакет, посылающая программа знать не может - в протоколе IPX такого подтверждения нет. Для того, чтобы точно знать, получен пакет или нет, получившая программа должна сама в ответ послать пакет, подтверждающий получение. Обе связывающиеся программы должны открыть свои сокеты. Перед выходом из программы долго живущие сокеты следует закрыть.

```
;Программа 1
;сетевая программа - сервер (принимает)
;сокет - 4001H, в перевернутом виде 1004H
DATA SEGMENT
      TEXT  DB 'Сокет не удалось открыть.',13,10,'$'
      TEXT1 DB 'Функция принятия пакета не выполнена',13,10,'$'
;заголовок пакета (30 байт)
CHECKSUM          DB 2 DUP(?)
LEN               DB 2 DUP(?)
TRANSPORTCONTROL  DB ?
PACKETTYPE        DB ?
DESTNETWORK       DB 4 DUP(?)
DESTNODE          DB 6 DUP(?)
DESTSOCKET        DB 2 DUP(?)
SOURCENETWORK     DB 4 DUP(?)
SOURCENODE        DB 6 DUP(?)
SOURCESOCKET      DB 2 DUP(?)
;здесь данные
STROKA  DB 32 DUP(?)
;здесь блок ECB
LINK              DB  4 DUP(?)
ESRADDRES         DB  4 DUP(0)
INUSE             DB  ?
CCODE             DB  ?
SOCKET            DW  1004H
IPXWORKSPACE      DB  4 DUP(?)
DRIVERWORKSPACE   DB 12 DUP(?)
```

```
IMMADDRESS          DB   6 DUP(?)
FRAGNENTCNT         DW   2
ADDRESS1            DW   OFFSET CHECKSUM
                    DW   SEG CHECKSUM
                    DW   30
ADDRESS2            DW   OFFSET STROKA
                    DW   SEG STROKA
SIZ                 DW   32
DATA ENDS
ST1 SEGMENT STACK 'STACK'
      DW 100 DUP(?)
ST1 ENDS
CODE SEGMENT
      ASSUME CS:CODE, DS:DATA, SS:ST1
 BEGIN:
      MOV AX,DATA
      MOV DS,AX
      CALL WHAT_POINT
      CMP  AL,0FFH
      JZ   OK1
      JMP  _END
OK1:
;открыть сокет данной программы
      MOV BX,0
      MOV AL,0FFH
      MOV DX,1004H
      CALL DWORD PTR CS:NET_POINT
      CMP AL,0
      JZ  OK_OPEN
;сокет не удалось открыть
      MOV AH,9
      LEA DX,TEXT
      INT 21H
      JMP SHORT _END
OK_OPEN:
;дать команду ожидания
      PUSH DS
      POP  ES
      LEA SI,LINK
      MOV BX,04H
;вызов функции ожидания
      CALL DWORD PTR CS:NET_POINT
      CMP  AL,0FFH
      JNZ  WAI
      MOV  AH, 9
```

```
        LEA   DX,TEXT1
        INT   21H
        JMP   SHORT _CLOSE
;цикл ожидания
WAI:
        MOV  AH,1
        INT   16H
        JNZ   _CLOSE
        CALL  FOR_WAIT
        CMP DS:INUSE,0
        JNZ  WAI
;пакет получен, выдать информацию
        MOV AH,9
        LEA  DX,STROKA
        INT  21H
;закрыть сокет
_CLOSE:
        MOV  BX,1
        MOV  DX,1004H
        CALL  DWORD  PTR  CS:NET_POINT
;выход из программы
_END:
        MOV  AH,4CH
        INT  21H
;раздел процедур
;определение точки входа в сетевой драйвер
WHAT_POINT  PROC
        MOV  AX,7A00H
        INT  2FH
        CMP  AL,0FFH
        JNZ  NO_IPX
        MOV  CS:NET_POINT,DI
        MOV  CS:NET_POINT+2,ES
 NO_IPX:
         RET
WHAT_POINT  ENDP
;процедура, вызываемая при ожидании
FOR_WAIT  PROC
        PUSH  BX
        MOV  BX,OAH
        CALL  DWORD  PTR  CS:NET_POINT
        POP  BX
        RET
FOR_WAIT  ENDP
```

```
;точка входа в сетевой драйвер
NET_POINT DW ?
      DW ?
CODE ENDS
      END BEGIN

;Программа 2
;сетевая программа - рабочая станция (посылает)
;сокет - 4002H, в перевернутом виде 2004H
DATA SEGMENT
      TEXT  DB 'Сокет не удалось открыть.',13,10, '$'
      TEXT1 DB 'Пакет отправлен.',13,10,'$'
;заголовок пакета
      CHECKSUM          DB 2 DUP(?)
      LEN               DB 2 DUP(?)
      TRANSPORTCONTROL  DB ?
      PACKETTYPE        DB 4
;номер сети 00001958H
      DESTNETWORK       DB 0
      DB 0
      DB 19H
      DB 58H
;номер рабочей станции 000C6C235427H, где находится
;программа 1
      DESTNODE          DB OOH
      DB OCHH
      DB 6CH
      DB 23H
      DB 54H
      DB 27H
;сокет 4001H - программы 1
      DESTSOCKET        DW 1004H
      SOURCENETWORK     DB 4 DUP(?)
      SOURCENODE        DB 6 DUP(?)
      SOURCESOCKET      DB 2 DUP(?)
;здесь данные - 32 байта
      STROKA             DB 'Эта строка передается по сети',13,10,'$'
;здесь блок ECB
      LINK              DB  4 DUP(?)
      ESRADDRES         DB  4 DUP(0)
      INUSE             DB  1
      CCODE             DB  ?
      SOCKET            DW  2004H
      IPXWORKSPACE      DB  4 DUP(?)
```

```
        DRIVERWORKSPACE     DB 12 DUP(?)
        IMMADDRESS          DB 00H
        DB 0C0H
        DB 6CH
        DB 23H
        DB 54H
        DB 27H
        FRAGNENTCNT         DW  2
        ADDRESS1            DW  OFFSET CHECKSUM
        DW  SEG CHECKSUM
        DW  30
        ADDRESS2            DW  OFFSET STROKA
        DW  SEG STROKA
        SIZ                 DW  32
DATA ENDS
ST1 SEGMENT STACK 'STACK'
        DW 100 DUP(?)
ST1 ENDS
CODE SEGMENT
        ASSUME CS:CODE, DS:DATA, SS:ST1
 BEGIN:
        MOV AX,DATA
        MOV DS,AX
        CALL WHAT_POINT
        CMP  AL,0FFH
        JZ   OK1
        JMP  _END
OK1:
;открыть сокет данной программы
        MOV BX,0
        MOV AL,0FFH
        MOV DX,2004H
        CALL DWORD PTR CS:NET_POINT
        CMP AL,0
        JZ  OK_OPEN
;не удалось открыть
        MOV AH,9
        LEA DX,TEXT
        INT 21H
        JMP SHORT _END
OK_OPEN:
;дать команду посылки пакета
        MOV BX,3
        PUSH DS
```

```
        POP   ES
        LEA   SI,DS:LINK
;вызов функции посылки
        CALL DWORD PTR CS:NET_POINT
;цикл ожидания
WAI:
        MOV  AH, 1
        INT  16H
        JNZ  _CLOSE
        CALL FOR_WAIT
        CMP DS:INUSE,0
        JNZ WAI
;пакет отправлен
        MOV AH,9
        LEA DX,TEXT1
        INT 21H
;закрыть сокет
_CLOSE:
        MOV BX,1
        MOV DX,2004H
        CALL DWORD PTR CS:NET_POINT
;выход из программы
_END:
        MOV AH,4CH
        INT 21H
;раздел процедур
;определение точки входа в сетевой драйвер
WHAT_POINT PROC
        MOV AX,7A00H
        INT 2FH
        CMP AL,OFFH
        JNZ NO_IPX
        MOV CS:NET_POINT,DI
        MOV CS:NET_POINT+2,ES
NO_IPX:
        RET
WHAT_POINT ENDP
/процедура, вызываемая при ожидании
FOR_WAIT PROC
        PUSH BX
        MOV BX,0AH
        CALL DWORD PTR CS:NET_POINT
        POP BX
        RET
FOR_WAIT ENDP
```

```
;точка входа в сетевой драйвер
NET_POINT  DW  ?
DW ?
CODE  ENDS
      END  BEGIN
```

Рис. 21.3. Пример двух программ, передающих и принимающих сообщение по протоколу IPX.

Ниже приведен еще один пример обмена данными по протоколу IPX. Представлены две программы NETS и NETW. Вначале программа NET W посылает данные, используя так называемый широковещательный адрес - FFFFFFFFFFFFH. Если использовать этот адрес, то пакет будет передаваться всем рабочим станциям данного сегмента. Программа NETS получает пакет. В блоке ESR при этом будет записан адрес рабочей станции, откуда пришел пакет. Таким образомм, программа NETS будет знать точный адрес программы NETW. Затем программа NETS уже по конкретному адресу посылает свои данные. После приема пакета в программе NETS стоит небольшая задержка. Это сделано для того, чтобы рабочая станция с программой NETW успела настроиться на прием. Такая пауза необходима только в том случае, если программа NETW запущена на медленном компьютере. Разумеется, более корректным был бы другой подход: на каждое посланное сообщение программа должна ждать подтверждения, т.е. программа NETS должна посылать свое сообщение до тех пор, пока не получит от программы NETW подтверждения. Мы, однако, не используем это, дабы не усложнить программы.

Заметим, что программа NETW осуществляет прием пакета посредством установки специальной процедуры прерывания - PROC_INT. Вызов этой процедуры осуществляет сетевой драйвер по приходу пакета.

```
;данная программа NETS вначале ждет прихода сообщения
;от программы NETW и узнает тем самым ее адрес
;затем по этому адресу посылает сообщение
;сокет программы - 4001H, в перевернутом виде 1004H
DATA  SEGMENT
      TEXT   DB 'Сокет не удалось открыть.',13,10,'$'
      TEXT1  DB 'Функция принятия пакета не выполнена',13,10,'$'
      TEXT2  DB 'Пакет отправлен.',13,10,'$'
;эта строка передается в ответ на полученное сообщение
      STROK  DB 'Строка, передаваемая обратно',13,10,'$'
;заголовок пакета
      CHECKSUM          DB 2 DUP(?)
      LEN               DB 2 DUP(?)
      TRANSPORTCONTROL  DB ?
      PACKETTYPE        DB 4
      DESTNETWORK       DB 0
      DB 0
```

```
        DB  19H
        DB  58H
        DESTNODE              DB 6 DUP(?)
        DESTSOCKET            DW 2004H
        SOURCENETWORK         DB 4 DUP(?)
        SOURCENODE            DB 6 DUP(?)
        SOURCESOCKET          DW 1004H
;здесь данные
STROKA   DB 32 DUP(?)
;здесь блок ECB
        LINK                  DB  4 DUP(?)
        ESRADDRES             DB  4 DUP(0)
        INUSE                 DB  ?
        CCODE                 DB  ?
        SOCKET                DW  1004H
        IPXWORKSPACE          DB  4 DUP(?)
        DRIVERWORKSPACE       DB 12 DUP(?)
        IMMADDRESS            DB  6 DUP(?)
        FRAGNENTCNT           DW  2
        ADDRESS1              DW  OFFSET CHECKSUM
        DW   SEG  CHECKSUM
        DW   30
        ADDRESS2              DW  OFFSET  STROKA
        DW   SEG  STROKA
        SIZ                   DW  32
DATA  ENDS
ST1  SEGMENT  STACK  'STACK'
        DW 100  DUP(?)
ST1 ENDS
CODE  SEGMENT
        ASSUME CS:CODE,  DS:DATA,  SS:ST1
 BEGIN:
        MOV  AX,DATA
        MOV  DS,AX
        CALL  WHAT_POINT
        CMP   AL,0FFH
        JZ    OK1
        JMP   _END
OK1:
;открыть сокет данной программы
        MOV  BX,0
        MOV  AL,0FFH
        MOV  DX,1004H
        CALL  DWORD  PTR  CS:NET_POINT
```

```
      CMP AL,0
      JZ  OK_OPEN
;сокет не удалось открыть
      MOV AH,9
      LEA DX,TEXT
      INT 21H
      JMP _END
OK_OPEN:
;дать команду ожидания
      PUSH DS
      POP  ES
      LEA SI,LINK
      MOV BX,04H
;вызов функции ожидания
      CALL DWORD PTR CS:NET_POINT
      CMP  AL,0FFH
      JNZ  WAI
      MOV  AH, 9
      LEA  DX,TEXT1
      INT  21H
      JMP  SHORT _CLOSE
;цикл ожидания
WAI:
      MOV  AH, 1
      INT  16H
      JNZ  _CLOSE
      CALL FOR_WAIT
      CMP DS:INUSE,0
      JNZ WAI
;пакет получен, выдать информацию
      MOV AH,9
      LEA DX,STROKA
      INT 21H
;установить адрес
      LEA  DI,DESTNODE
      LEA SI,IMMADDRESS
      MOV CX,6
LOOQ:
      LODSB
      STOSB
      LOOP LOOQ
      MOV  DESTSOCKET,2004H
      MOV  SOCKET,1004H
      MOV  SOURCESOCKET,1004H
```

```
;переслать строку, которая будет пересылаться обратно
      LEA  SI,STROK
      LEA  DI,STROKA
KLL:
      LODSB
      CMP  AL, '$'
      JZ   PAU
      STOSB
      JMP  SHORT KLL
;здесь пауза, чтобы программа NETW настроилась на прием
PAU:
      CALL PAUSE
;дать команду посылки пакета
      MOV BX,3
      PUSH DS
      POP  ES
      LEA  SI,DS:LINK
;вызов функции посылки
      CALL DWORD PTR CS:NET_POINT
;цикл ожидания
WAI1:
      MOV  AH,1
      INT  16H
      JNZ  _CLOSE
      CALL FOR_WAIT
      CMP DS:INUSE,0
      JNZ WAI1
;пакет отправлен
      MOV AH,9
      LEA DX,TEXT2
      INT 21H
;закрыть сокет
_CLOSE:
      MOV BX,1
      MOV  DX,1004H
      CALL DWORD PTR CS:NET_POINT
;выход из программы
_END:
      MOV AH,4CH
      INT 21H
;раздел процедур
;определение точки входа в сетевой драйвер
WHAT_POINT PROC
      MOV AX,7A00H
      INT 2FH
```

```
        CMP AL,0FFH
        JNZ  NO_IPX
        MOV CS:NET_POINT,DI
        MOV CS:NET_POINT+2,ES
NO_IPX:
        RET
WHAT_POINT ENDP
;процедура, вызываемая при ожидании
FOR_WAIT PROC
        PUSH BX
        MOV BX,0AH
        CALL DWORD PTR CS:NET_POINT
        POP BX
        RET
FOR_WAIT ENDP
PAUSE  PROC
        PUSH CX
        MOV   CX,20
DAL:
        PUSH CX
        MOV   CX,0FFFFH
PA:     LOOP PA
        POP   CX
        LOOP DAL
        POP   CX
        RET
PAUSE ENDP
;точка входа в сетевой драйвер
NET_POINT DW ?
        DW ?
CODE ENDS
        END BEGIN

;программа NETW, посылает вначале на все рабочие
;станции сообщение, это сообщение должна принять
/и программа NETS
/затем программа ждет ответного сообщения
;сокет - 4002H, в перевернутом виде 2004H
DATA SEGMENT
        TEXT  DB 'Сокет не удалось открыть. ',13,10, '$'
        TEXT1 DB 'Пакет отправлен.', 13,10,'$'
        TEXT2 DB 'Функция принятия пакета не выполнена',13,10,'$'
/заголовок пакета
        CHECKSUM            DB 2 DUP(?)
        LEN                 DB 2 DUP(?)
```

```
        TRANS PORTCONTROL    DB  ?
        PACKETTYPE           DB  4
        DESTNETWORK          DB  58H
        DB  19H
        DB  0
        DB  0
;широковещательный  адрес
        DESTNODE             DB  OFFH
        DB  O'FFH
        DB  OFFH
        DB  OFFH
        DB  OFFH
        DB  OFFH
        DESTSOCKET           DW  1004H
        SOURCENETWORK        DB  4  DUP(?)
        SOURCENODE           DB  6  DUP(?)
        SOURCESOCKET         DW  2004H
;здесь  данные
        STROKA                DB  'Эта строка передается по сети',13,10,'$'
;здесь  блок  ECB
        LINK                 DB   4  DUP(?)
        ESRADDRES            DB   4  DUP(0)
        INUSE                DB   1
        CCODE                DB   7
        SOCKET               DW   2004H
        IPXWORKSPACE         DB   4  DUP(?)
        DRIVERWORKSPACE      DB  12  DUP(?)
        IMMADDRESS           DB  OFFH
        DB  OFFH
        DB  OFFH
        DB  OFFH
        DB  OFFH
        DB  OFFH
        FRAGNENTCNT          DW   2
        ADDRESS1             DW   OFFSET  CHECKSUM
        DW   SEG  CHECKSUM
        DW   30
        ADDRESS2             DW   OFFSET  STROKA
        DW   SEG  STROKA
        SIZ                  DW   32
DATA  ENDS
ST1  SEGMENT  STACK  'STACK'
        DW  100  DUP(?)
ST1  ENDS
```

```
CODE  SEGMENT
      ASSUME CS:CODE, DS:DATA, SS:ST1
BEGIN:
      MOV AX,DATA
      MOV DS,AX
      CALL WHAT_POINT
      CMP  AL,0FFH
      JZ   OK1
      JMP  _END
OK1:
;открыть сокет
;данной программы
      MOV BX,0
      MOV AL,0FFH
      MOV DX,2004H
      CALL  DWORD PTR CS:NET_POINT
      CMP AL,0
      JZ   OK_OPEN
;не удалось открыть
      MOV AH, 9
      LEA  DX,TEXT
      INT  21H
      JMP SHORT _END
OK_OPEN:
;дать команду посылки пакета
;пакет будет передаваться на все станции
      MOV  BX,3
      PUSH DS
      POP  ES
      LEA   SI,DS:LINK
;вызов функции посылки
      CALL  DWORD PTR CS:NET_POINT
;цикл ожидания
WAI:
      MOV  AH, 1
      INT  16H
      JNZ  _CLOSE
      CALL  FOR_WAIT
      CMP  DS:INUSE,0
      JNZ WAI
;пакет отправлен
      MOV  AH,9
      LEA DX,TEXT1
      INT  21H
```

```
/теперь ждем прихода пакета уже по конкретному адресу,
;т.е. по адресу, по которому находится данная программа
;установим адрес процедуры обработки прерывания
      PUSH CS
      POP  BX
      MOV  WORD PTR ESRADDRES+2,BX
      LEA  BX,PROC_INT
      MOV  WORD PTR ESRADDRES,BX
;дать команду ожидания
      PUSH DS
      POP  ES
      LEA  SI,LINK
      MOV  BX,04H
;вызов функции ожидания
      CALL DWORD PTR CS:NET_POINT
      CMP  AL,0FFH
      JNZ  WAI1
      MOV  AH, 9
      LEA  DX,TEXT1
      INT  21H
      JMP  SHORT _CLOSE
;цикл ожидания
WAI1:
;ждем прихода пакета либо нажатия клавиши
      MOV  AH, O
      INT  16H
;закрыть сокет
_CLOSE:
      MOV  BX,1
      MOV  DX,2004H
      CALL DWORD PTR CS:NET_POINT
;выход из программы
_END:
      MOV  AH,4CH
      INT  21H
;раздел процедур
;определение точки входа в сетевой драйвер
      WHAT_POINT PROC
      MOV  AX,7A00H
      INT  2FH
      CMP  AL,0FFH
      JNZ  NO_IPX
      MOV  CS:NET_POINT,DI
      MOV  CS:NET_POINT+2,ES
```

```
NO_IPX:
       RET
WHAT_POINT ENDP
;процедура, вызываемая при ожидании
FOR_WAIT PROC
       PUSH BX
       MOV BX,0AH
       CALL DWORD PTR CS:NET_POINT
       POP BX
       RET
FOR_WAIT ENDP
;процедура обработки ситуации прихода пакета
PROC_INT PROC
;пакет получен, выдать информацию
       MOV AH,9
       LEA DX,STROKA
       INT 21H
;послать в буфер клавиатуры символ
       MOV AH,5
       MOV CL,32
       INT 16H
       RETF
PROC_INT ENDP
;точка входа в сетевой драйвер
NET_POINT DW ?
       DW ?
CODE ENDS
       END BEGIN
```

Рис. 21.4. Еще две программы, связывающиеся друг с другом по сети.

Диагностика сети.

Для успешной работы Вашей сетевой программы она должна уметь получать информацию о сети. Особенно это важно в сложных сетях, содержащих несколько сегментов. Один из способов получения информации — это использование специального диагностического сокета - 456H. Идея заключается в следующем: программа должна послать специальный запрос по широковещательному адресу (FFFFFFFFFFFFH), а затем ждать ответа от всех станций. Анализируя приходящую информацию, программа определяет сетевые адреса мостов и номера подключенных к ним сетей. Посылая в эти сети запросы, можно получить и их конфигурацию. Двигаясь таким образом, можно узнать конфигурацию всей сети.

Для посылки запроса следует сформировать обычный заголовок в 30 байт. Блок данных имеет следующую структуру:

```
Exclusions db ?
list1      db 6 dup(?)
. .
list80     db 6 dup(?)
```

В заголовке пакета может стоять адрес конкретной станции. Тогда информация будет получена только от нее. Если указан широковещательный адрес, то в блоке данных можно указать адреса станций, от которых не требуется информации.

В блоке ECB ImmAddres также указывается широковещательный адрес, а в дальнейшем адреса мостов. Сокет 456H не нужно открывать или закрывать. Он используется для формирования адреса. Для передачи же запроса и приема ответа следует открыть свой сокет.

Пришедший пакет состоит из стандартного заголовка и блока данных. Блок данных состоит из двух частей.

Первая часть:

```
DB ? ; старшая часть версии диагностического сервиса
DB ? ; младшая часть версии диагностического сервиса
DW ? ; номер SPX-сокета для диагностики
DB ? ; количество компонентов программного и аппаратного
     ; обеспечения, информация о которых содержится в
     ; данном пакете
```

Вторая часть.

Состоит из отдельных компонент. Компонента начинается с байта-идентификатора. Этот байт говорит нам о том, какая структура далее будет использована: простая или расширенная. Простая структура состоит всего из одного байта.

Значение этого байта:

0 - драйвер IPX/SPX,

1 - драйвер программного обеспечения моста,

2 - драйвер сетевой оболочки рабочей станции,

3 - сетевая оболочка,

4 - сетевая оболочка в виде VAP-процесса.

Расширенная структура сама по себе имеет две части: фиксированную и переменной длины.

Вот структура фиксированной части:

```
DB ? ;идентификатор
DB ? ;количество сетей, подключенных к мосту (ид.=5)
```

Значение идентификатора: 5 - внешний мост, 6 - файл-сервер с внутренним мостом, 7 - невыделенный файл-сервер

Переменная часть описывает сети, подключенные к мосту (ид.=5). Вот структура, описывающая сеть:

```
DB ? ;тип сети
DB 4 DUP(?) ;номер сети
DB 6 DUP(?) ;сетевой адрес адаптера
```

Типы сетей:

0 - сеть, к которой подключен сетевой адаптер

1 - сеть с виртуальным сетевым адаптером (невыделенный файл-сервер)

2 - переназначенная удаленная линия (связь через модем).

Пример программы, которая тестирует локальную сеть указанным средством, Вы найдете в книге [31].

Хочу заметить, что диагностические средства IPX могут понадобиться Вам при создании сетевых приложений, обменивающихся информацией друг с другом. Дело в том, что сеть может состоять из нескольких сегментов, которые соединяются друг с другом мостами. Ваша программа должна найти программу, с которой она хочет связаться в этой сложной системе. Ничего не поделаешь, но придется выяснять адреса мостов, затем посылать диагностические пакеты через эти мосты и т.д. Алгоритм поиска напоминает поиск файла по дереву каталогов, т.е. рекурсивен.

IV. Протокол SPX.

Протокол SPX (Sequenced Packet Exchange) является протоколом более высокого уровня, чем IPX. В своей работе он использует средства протокола IPX. Протокол SPX дает более надежный способ обмена информацией, так как в основе его лежит установка канала между двумя программами, работающими в сети. Можно сказать, что посредством протокола SPX осуществляется синхронная связь (асинхронная в случае протокола IPX).

Формат пакета SPX совпадает с форматом пакета IPX, но в конце добавляется еще 12 байт, то есть длина заголовка пакета становится равной 42 байта. Ниже представлен формат последних 12 байт.

Длина	Название	Предназначение
1	CONNCONTROL	набор битовых флагов 01-08H - зарезервировано ЮН - бит может использоваться программой для сигнализации об окончании передачи данных 20H - игнорируется SPX 40H - используется драйвером SPX 80H - используется драйвером SPX
1	DATASTREAMTYPE	Набор битовых флагов OOH-FDH - игнорируются IPX, FEH - команда завершения связи посылает пакет с установленным данными битами, FFH - системный пакет, подтверждающий завершение связи
2	SOURCECONNID	номер канала связи передающей программы
2	DESTCONNID	номер канала связи принимающей стороны
2	SEQNUMBER	счетчик пакетов, переданных по каналу в одном направлении
2	ACKNUMBER	номер следующего пакета, который должен быть принят драйвером SPX
2	ALLOCNUMBER	количество буферов, распределенных для приема пакета.

Заметим, что блок ECB в протоколе SPX имеет ту же структуру, что и для протокола IPX.

Сетевые функции SPX.

Функция разрыва соединения.
Вход:
BX=14H
DX - номер соединения.

Инициализация SPX.
Вход:
BX=10H
AL=0
Выход:
AL=0 SPX не инсталлирован
=FFH SPX инсталлирован
BH-старшая часть номера версии SPX
BL-младшая часть номера версии SPX
CX-максимальное число соединений
DX-доступное число соединений.

Создать соединение между исходной станцией и заданной станцией назначения.
Вход:
BX=11H
AL-число попыток
AH-флаг сканирования соединения
ES:SI - указатель на ECB
Выход:
AL - код завершения
O - успешно (SPX пытается создать соединение)
EFH - локальная таблица соединений полна
FDH - ошибка в ECB
FFH - посылающий сокет не открыт

Окончательный код завершения:
00 - соединение установлено
EDH- нет ответа из узла назначения
EFH- таблица соединений полна
FCH- отправляющий сокет закрыт во время фоновой обработки
FDH- ошибка в ECB
FFH- посылающее гнездо не открыто.

Функция возвращения статуса существующего соединения.
Вход:
BX=15H
DX-идентификатор соединения
ES:SI - указатель на буфер ответа
Выход:
AL-00 успешный
EEH нет такого соединения

Функция ожидания соединения.
Вход:
BX=12H
AL-число попыток
AH-флаг сканирования соединения.
ES:SI - указатель на ECB
Выход:
окончательный код завершения
00 - соединение установлено
EFH - локальная таблица соединений полна
FCH - снято ожидание
FFH - сокет не открыт

Функция ожидания пакета.
Вход:
BX=17H
ES:SI - указатель на ECB.
Выход:
окончательный код завершения
00 - успешно
EDH - соединение разрушено
FCH - соединение снято
FDH - буфер очень мал
FFH - сокет не открыт

Функция посылки пакета.
Вход:
BX=16H
DX-номер соединения
ES:SI - указатель на ECB
Выход:
окончательный код завершения
00 - успешно
ECH - другая станция закрыла соединение
EDH - соединение разрушено
EEH - нет такого соединения
FCH - сокет не открыт
FDH - пакет слишком большой

Функция разрыва соединения.
Вход:
BX=13H
DX - номер соединения
ES:SI - указатель на ECB
Выход:
окончательный код завершения
00 - успешно

ECH - другая станция закрыла соединение
EDH - соединение разрушено
EEH - нет такого соединения
FCH - сокет не открыт
FDH - буфер очень большой

Ниже приводятся две программы, устанавливающие между собой канал SPX связи. Программы не обмениваются пакетами – я считаю, что читатель сможет самостоятельно добавить соответствующие процедуры.

Обращаю Ваше внимание на пятикратный вызов функции приема пакета (функция 17H) перед установкой связи. Для нашего случая в принципе достаточно было бы и одинарного вызова. Не считая себя глубоким специалистом в области механизмов работы драйвера протокола SPX, я лишь сошлюсь на [30,31] и приведу цитату из [30]: "Выполните 5 вызовов функции 17H, чтобы передать SPX пул блоков ECB и буферов пакетов. SPX использует несколько из этих внутренним образом прозрачно для вашей программы."

```
;программа, использующая протокол SPX (1)
;программа работает на сокете 7001H (1007H)
;ждет соединения
DATA SEGMENT
;пять пакетов и блоков ECB для вызова функции
;приема пакета
;*****************************************
;заголовок пакета 1
 CHECKSUM           DB 2 DUP(?)
 LEN                DB 2 DUP(?)
 TRANSPORTCONTROL   DB 0
 PACKETTYPE         DB 5
 DESTNETWORK        DB 4 DUP(?)
 DESTNODE           DB 6 DUP(?)
 DESTSOCKET         DB 2 DUP(?)
 SOURCENETWORK      DB 4 DUP(?)
 SOURCENODE         DB 6 DUP(?)
 SOURCESOCKET       DW 1007H
;SPX
 CONNCONTROL        DB ?
 DATASTREAMTYPE     DB ?
 SOURCECONNID       DW ?
 DESTCONNID         DW ?
 SEQNUMBER          DW ?
 ACKNUMBER          DW ?
 ALLOCNUMBER        DW ?
```

```
;здесь данные
STROKA   DB  534  DUP(?)
;блок ECB  1
LINK1                 DB   4 DUP(?)
ESRADDRES1            DB   4 DUP(0)
INUSE1                DB   ?
CCODE1                DB   ?
SOCKET1               DW   1007H
IPXWORKSPACE1         DB   4 DUP(?)
DRIVERWORKSPACE1      DB  12 DUP(?)
IMMADDRESS1           DB   6 DUP(?)
FRAGNENTCNT1          DW   2
ADDRESS11             DW   OFFSET  CHECKSUM
       DW   SEG CHECKSUM
       DW   42
       ADDRESS21             DW   OFFSET  STROKA
       DW   SEG  STROKA
       SIZ1                  DW   534
;****************************************
;заголовок пакета 3
       CHECKSUM3             DB 2  DUP(?)
       LEN3                  DB 2  DUP(?)
       TRANSPORTCONTROL3     DB 0
       PACKETTYPE3           DB 5
       DESTNETWORK3          DB 4  DUP(?)
       DESTNODE3             DB 6  DUP(?)
       DESTSOCKET3           DB 2  DUP(?)
       SOURCENETWORK3        DB 4  DUP(?)
       SOURCENODE3           DB 6  DUP(?)
       SOURCESOCKET3         DW 1007H
;SPX
       CONNCONTROL3          DB ?
       DATASTREAMTYPE3       DB ?
       SOURCECONNID3         DW ?
       DESTCONNID3           DW ?
       SEQNUMBER3            DW ?
       ACKNUMBER3            DW ?
       ALLOCNUMBER3          DW ?
;здесь данные
STROKA3  DB  534  DUP(?)
;блок ECB  3
       LINK3                 DB   4  DUP(?)
       ESRADDRES3            DB   4  DUP(0)
       INUSE3                DB   ?
```

```
      CCODE3              DB  ?
      SOCKET3             DW  1007H
      IPXWORKSPACE3       DB  4 DUP(?)
      DRIVERWORKSPACE3    DB 12 DUP(?)
      IMMADDRESS3         DB  6 DUP(?)
      FRAGNENTCNT3        DW  2
      ADDRESS13           DW  OFFSET CHECKSUM3
      DW  SEG CHECKSUM3
      DW  42
      ADDRESS23           DW  OFFSET STROKA3
      DW  SEG STROKA3
      SIZ3                DW  534
;*******************************************
;заголовок пакета 4
      CHECKSUM4           DB 2 DUP(?)
      LEN4                DB 2 DUP(?)
      TRANSPORTCONTROL4   DB 0
      PACKETTYPE4         DB 5
      DESTNETWORK4        DB 4 DUP(?)
      DESTNODE4           DB 6 DUP(?)
      DESTSOCKET4         DB 2 DUP(?)
      SOURCENETWORK4      DB 4 DUP(?)
      SOURCENODE4         DB 6 DUP(?)
      SOURCESOCKET4       DW 1007H
;SPX
      CONNCONTROL4        DB ?
      DATASTREAMTYPE4     DB ?
      SOURCECONNID4       DW ?
      DESTCONNID4         DW ?
      SEQNUMBER4          DW ?
      ACKNUMBER4          DW ?
      ALLOCNUMBER4        DW ?
;здесь данные
STROKA4 DB 534 DUP(?)
;блок ECB 4
      LINK4               DB  4 DUP(?)
      ESRADDRES4          DB  4 DUP(0)
      INUSE4              DB  ?
      CCODE4              DB  ?
      SOCKET4             DW  1007H
      IPXWORKSPACE4       DB  4 DUP(?)
      DRIVERWORKSPACE4    DB 12 DUP{?)
      IMMADDRESS4         DB  6 DUP(?)
      FRAGNENTCNT4        DW  2
```

```
      ADDRESS14          DW   OFFSET  CHECKSUM4
      DW   SEG  CHECKSUM4
      DW   42
      ADDRESS24          DW   OFFSET  STROKA4
      DW   SEG  STROKA4
      SIZ4               DW   534
;*****************************************
;заголовок пакета 5
      CHECKSUM5          DB 2  DUP(?)
      LEN5               DB 2  DUP(?)
      TRANSPORTCONTROL5  DB 0
      PACKETTYPE5        DB 5
      DESTNETWORK5       DB 4  DUP(?)
      DESTNODE5          DB 6  DUP(?)
      DESTSOCKET5        DB 2  DUP(?)
      SOURCENETWORK5     DB 4  DUP(?)
      SOURCENODE5        DB 6  DUP(?)
      SOURCESOCKET5      DW 1007H
;SPX
      CONNCONTROL5       DB ?
      DATASTREAMTYPE5    DB ?
      SOURCECONNID5      DW ?
      DESTCONNID5        DW ?
      SEQNUMBER5         DW ?
      ACKNUMBER5         DW ?
      ALLOCNUMBER5       DW ?
;здесь данные
STROKA5 DB 534  DUP(?)
;блок ECB  5
      LINK5              DB   4  DUP(?)
      ESRADDRES5         DB   4  DUP(O)
      INUSE5             DB   ?
      CCODE5             DB   ?
      SOCKET5            DW   1007H
      IPXWORKSPACE5      DB   4  DUP(?)
      DRIVERWORKSPACE5   DB  12  DUP(?)
      IMMADDRESS5        DB   6  DUP(?)
      FRAGNENTCNT5       DW   2
      ADDRESS15          DW   OFFSET  CHECKSUM5
      DW   SEG  CHECKSUM5
      DW   42
      ADDRESS25          DW   OFFSET  STROKA5
      DW   SEG  STROKA5
      SIZ5               DW   534
```

```
;****************************************
;заголовок пакета 6
      CHECKSUM6          DB 2 DUP(?)
      LEN6               DB 2 DUP(?)
      TRANSPORTCONTROL6  DB 0
      PACKETTYPE6        DB 5
      DESTNETWORK6       DB 4 DUP(?)
      DESTNODE6          DB 6 DUP(?)
      DESTSOCKET6        DB 2 DUP(?)
      SOURCENETWORK6     DB 4 DUP(?)
      SOURCENODE6        DB 6 DUP(?)
      SOURCESOCKET6      DW 1007H
;SPX
      CONNCONTROL6       DB 7
      DATASTREAMTYPE6    DB ?
      SOURCECONNID6      DW 7
      DESTCONNID6        DW 7
      SEQNUMBER6         DW 7
      ACKNUMBER6         DW 7
      ALLOCNUMBER6       DW 7
;здесь данные
STROKA6 DB 534 DUP(?)
;блок ECB 6
      LINK6              DB  4 DUP(?)
      ESRADDRES6         DB  4 DUP(0)
      INUSE6             DB  7
      CCODE6             DB  7
      SOCKET6            DW  1007H
      IPXWORKSPACE6      DB  4 DUP(?)
      DRIVERWORKSPACE6   DB 12 DUP(?)
      IMMADDRESS6        DB  6 DUP(?)
      FRAGNENTCNT6       DW  2
      ADDRESS16          DW  OFFSET CHECKSUM6
      DW  SEG CHECKSUM6
      DW  42
      ADDRESS26          DW  OFFSET STROKA6
      DW  SEG STROKA6
      SIZ6               DW  534
;пакет и блок ECB для установления связи
;*////************************\\\************
; заголовок пакета 2
      CHECKSUM2          DB 2 DUP(?)
      LEN2               DB 2 DUP(?)
      TRANSPORTCONTROL2  DB 0
```

```
       PACKETTYPE2          DB 5
       DESTNETWORK2         DB 4 DUP (?)
       DESTNODE2            DB 6 DUP (?)
       DESTSOCKET2          DB 2 DUP (?)
       SOURCENETWORK2       DB 4 DUP (?)
       SOURCENODE2          DB 6 DUP (?)
       SOURCESOCKET2        DW 1007H
;SPX
       CONNCONTROL2         DB 0
       DATASTREAMTYPE2      DB 0
       SOURCECONNID2        DW 0
       DESTCONNID2          DW 0
       SEQNUMBER2           DW 0
       ACKNUMBER2           DW 0
       ALLOCNUMBER2         DW 0
;здесь данные
STROKA2                    DB 534 DUP(?)
;блок ECB 1
       LINK                 DB  4 DUP(?)
       ESRADDRES            DB  4 DUP(0)
       INUSE                DB  ?
       CCODE                DB  ?
       SOCKET               DW  1007H
       IPXWORKSPACE         DB  4 DUP(?)
       DRIVERWORKSPACE      DB 12 DUP(?)
       IMMADDRESS           DB  6 DUP(?)
       FRAGNENTCNT          DW  1
       ADDRESS1             DW  OFFSET CHECKSUM2
       DW   SEG CHECKSUM2
       DW   42
       ADDRESS2             DW  OFFSET STROKA2
       DW   SEG STROKA2
       SIZ                  DW  534
;область сообщений
TEXT1 DB "Сетевые функции отсутствуют.",13,10,'$'
TEXT2 DB "Сокет не удалось открыть.",13,10,'$'
TEXT3 DB "Канал установлен.",13,10,'$'
        _DX DW ?
DATA ENDS
STAC SEGMENT STACK
       DW 100 DUP(?)
STAC ENDS
CODE SEGMENT
       ASSUME CS:CODE, DS:DATA, SS:STAC, ES:DATA
```

```
BEGIN:
       MOV   AX,DATA
       MOV   DS,AX
       CALL  WHAT_POINT
       CMP AL,0FFH
       JZ    _OK1
       LEA DX,TEXT1
       MOV AH,9
       INT  21H
       JMP _END
_OK1:
;протокол  SPX  присутствует
;открыть  сокет
       XOR   BX,BX
       MOV   AL,0H
       MOV   DX,1007H
       CALL  DWORD  PTR  CS:NET_POINT
       CMP   AL,0
       JZ     OK_OPEN
;сокет не удалось открыть
       LEA  DX,TEXT2
       MOV AH,9
       INT  21H
       JMP _END
OK_OPEN:
;сокет открыт
       PUSH DS
       POP   ES
;пятикратный вызов функции приема пакета  (17H)
LEA  SI,DS:LINK1
       MOV  BX,17H
       CALL  DWORD  PTR  CS:NET_POINT
       LEA  SI,DS:LINK3
       MOV  BX,17H
       CALL  DWORD  PTR  CS:NET_POINT
       LEA  SI,DS:LINK4
       MOV  BX,17H
       CALL  DWORD  PTR  CS:NET_POINT
       LEA  SI,DS:LINK5
       MOV  BX,17H
       CALL  DWORD  PTR  CS:NET_POINT
       LEA  SI,DS:LINK6
       MOV  BX,17H
       CALL  DWORD  PTR  CS:NET_POINT
;---------------------------------------
```

```
;ждать соединения
        MOV BX,12H
        MOV AL,0
        MOV AH,0
        LEA SI,ES:LINK
        CALL DWORD PTR CS:NET_POINT
;ждем установления канала
WAI:
        MOV AH,1
        INT 16H
        JNZ _CLOSE
        CALL FOR_WAIT
        CMP DS:INUSE,0
        JNZ WAI
        MOV _DX,DX
;канал установлен
        LEA DX,TEXT3
        MOV AH,9
        INT 21H
_ABORT:
        MOV DX,_DX
        MOV BX,14H
        CALL DWORD PTR CS:NET_POINT
;разрываем канал
_CLOSE:
        MOV BX,1
        MOV DX,1007H
        CALL DWORD PTR CS:NET_POINT
_END:
        MOV AX,4C00H
        INT 21H
;область процедур
;определение наличия сетевых протоколов и
;определение точки входа в сетевой драйвер
WHAT_POINT PROC
        MOV AX,7A00H
        INT 2FH
        CMP AL,0FFH
        JNZ NO_IPX_SPX
        MOV CS:NET_POINT,DI
        MOV CS:NET_POINT+2,ES
;проверяем наличие SPX
        XOR AL,AL
        MOV BX,10H
        CALL DWORD PTR CS:NET_POINT
```

```
NO_IPX_SPX:
      RETN
WHAT_POINT ENDP
;процедура, вызываемая при ожидании
FOR_WAIT PROC
      PUSH BX
      MOV BX,0AH
      CALL DWORD PTR CS:NET_POINT
      POP BX
      RET
FOR_WAIT ENDP
PAUSE PROC
      PUSH CX
      MOV  CX,20
DAL:
      PUSH CX
      MOV  CX,0FFFFH
PA:    LOOP PA
      POP  CX
      LOOP DAL
      POP  CX
      RET
PAUSE ENDP
;адрес вызова сетевых процедур
NET_POINT  DW ?
      DW ?
CODE ENDS
      END BEGIN
```

```
;программа, использующая протокол SPX  (2);программа работает
на сокете ;7002H(2007H)
;устанавливает соединение
DATA SEGMENT
;пять пакетов и блоков ECB для вызова функции
;приема пакета
;************************************
;заголовок пакета 1
CHECKSUM              DB 2 DUP(?)
LEN                   DB 2 DUP(?)
TRANSPORTCONTROL      DB 0
PACKETTYPE            DB 5
DESTNETWORK           DB 4 DUP(0)
DESTNODE              DB 6 DUP(0)
```

```
DESTSOCKET          DW  0
SOURCENETWORK       DB  4 DUP(?)
SOURCENODE          DB  6 DUP(?)
SOURCESOCKET        DB  2 DUP(?)
;SPX
CONNCONTROL         DB  7
DATASTREAMTYPE      DB  7
SOURCECONNID        DW  7
DESTCONNID          DW  7
SEQNUMBER           DW  7
ACKNUMBER           DW  7
ALLOCNUMBER         DW  7
;здесь данные
STROKA              DB  534 DUP(?)
;************************************
;здесь блок ECB 1
LINK1               DB   4 DUP(?)
ESRADDRES1          DB   4 DUP(0)
INUSE1              DB   1
CCODE1              DB   7
SOCKET1             DW   2007H
IPXWORKSPACE1       DB   4 DUP(?)
DRIVERWORKSPACE1    DB  12 DUP(?)
IMMADDRESS1         DB  6 DUP(0)
FRAGNENTCNT1        DW  2
ADDRESS11           DW   OFFSET CHECKSUM
                    DW   SEG CHECKSUM
                    DW   42
ADDRESS21           DW   OFFSET STROKA
                    DW   SEG STROKA
SIZ1                DW   534
;*********************************
;заголовок пакета 3
CHECKSUM3           DB  2 DUP(?)
LEN3                DB  2 DUP(?)
TRANSPORTCONTROL3   DB  0
PACKETTYPE3         DB  5
DESTNETWORK3        DB  4 DUP(O)
DESTNODE3           DB  6 DUP(0)
DESTSOCKET3         DW  0
SOURCENETWORK3      DB  4 DUP(?)
SOURCENODE3         DB  6 DUP(?)
SOURCESOCKET3       DB  2 DUP(?)
;SPX
```

```
CONNCONTROL3        DB  7
DATASTREAMTYPE3     DB  7
SOURCECONNID3       DW  ?
DESTCONNID3         DW  7
SEQNUMBER3          DW  7
ACKNUMBER3          DW  7
ALLOCNUMBER3        DW  7
;здесь данные
STROKA3             DB  534 DUP(?)
;***********************************
;здесь блок ECB 3
LINK3               DB   4 DUP(?)
ESRADDRES3          DB   4 DUP(0)
INUSE3              DB   1
CCODE3              DB   7
SOCKET3             DW   2007H
IPXWORKSPACE3       DB   4 DUP(?)
DRIVERWORKSPACE3    DB  12 DUP(?)
IMMADDRESS3         DB  6 DUP (0)
FRAGNENTCNT3        DW  2
ADDRESS31           DW   OFFSET CHECKSUM3
                    DW   SEG CHECKSUM3
                    DW   42
ADDRESS32           DW   OFFSET STROKA3
                    DW   SEG STROKA3
SIZ3                DW   534
;***********************************
;заголовок пакета 4
CHECKSUM4           DB  2 DUP (?)
LEN4                DB  2 DUP (?)
TRANSPORTCONTROL4   DB  0
PACKETTYPE4         DB  5
DESTNETWORK4        DB  4 DUP (0)
DESTNODE4           DB  6 DUP (0)
DESTSOCKET4         DW  0
SOURCENETWORK4      DB  4 DUP (?)
SOURCENODE4         DB  6 DUP (?)
SOURCESOCKET4       DB  2 DUP (?)
;SPX
CONNCONTROL4        DB  7
DATASTREAMTYPE4     DB  7
SOURCECONNID4       DW  7
DESTCONNID4         DW  7
SEQNUMBER4          DW  ?
```

```
ACKNUMBER4          DW ?
ALLOCNUMBER4        DW ?
;здесь данные
STROKA4             DB 534 DUP(?)
;***********************************
;здесь блок ECB 4
LINK4               DB  4 DUP(?)
ESRADDRES4          DB  4 DUP(0)
INUSE4              DB  1
CCODE4              DB  ?
SOCKET4             DW  2007H
IPXWORKSPACE4       DB  4  DUP(?)
DRIVERWORKSPACE4    DB 12  DUP(?)
IMMADDRESS4         DB 6 DUP(0)
FRAGNENTCNT4        DW 2
ADDRESS41           DW  OFFSET CHECKSUM4
                    DW  SEG CHECKSUM4
                    DW  42
ADDRESS42           DW  OFFSET STROKA4
                    DW  SEG STROKA4
SIZ4                DW  534
;***********************************
;заголовок пакета 5
CHECKSUM5           DB 2 DUP(?)
LEN5                DB 2 DUP(?)
TRANSPORTCONTROL5   DB 0
PACKETTYPE5         DB 5
DESTNETWORK5        DB 4 DUP(0)
DESTNODE5           DB 6 DUP(0)
DESTSOCKET5         DW 0
SOURCENETWORK5      DB 4 DUP(?)
SOURCENODE5         DB 6 DUP(?)
SOURCESOCKET5       DB 2 DUP(?)
;SPX
CONNCONTROL5        DB ?
DATASTREAMTYPE5     DB ?
SOURCECONNID5       DW ?
DESTCONNID5         DW ?
SEQNUMBER5          DW ?
ACKNUMBER5          DW ?
ALLOCNUMBER5        DW ?
;здесь данные
STROKA5             DB 534 DUP(?)
;***********************************
```

```
;здесь блок ECB 5
LINK5               DB  4 DUP(?)
ESRADDRES5          DB  4 DUP(0)
INUSE5              DB  1
CCODE5              DB  ?
SOCKET5             DW  2007H
IPXWORKSPACE5       DB  4 DUP(?)
DRIVERWORKSPACE5    DB 12 DUP(?)
IMMADDRESS5         DB 6 DUP(0)
FRAGNENTCNT5        DW 2
ADDRESS51           DW  OFFSET CHECKSUM5
                    DW  SEG CHECKSUM5
                    DW  42
ADDRESS52           DW  OFFSET STROKA5
                    DW  SEG STROKA5
SIZ5                DW  534
;************************************
;заголовок пакета 6
CHECKSUM6           DB 2 DUP(?)
LEN6                DB 2 DUP(?)
TRANSPORTCONTROL6 DB 0
PACKETTYPE6         DB 5
DESTNETWORK6      DB 4 DUP(O)
DESTNODE6           DB 6 DUP(0)
DESTSOCKET6         DW 0
SOURCENETWORK6      DB 4 DUP(?)
SOURCENODE6         DB 6 DUP(?)
SOURCESOCKET6       DB 2 DUP(?)
;SPX
CONNCONTROL6        DB ?
DATASTREAMTYPE6     DB ?
SOURCECONNID6       DW ?
DESTCONNID6         DW ?
SEQNUMBER6          DW ?
ACKNUMBER6          DW ?
ALLOCNUMBER6        DW ?
;здесь данные
STROKA6             DB 534 DUP(?)
;************************************
;здесь блок ECB 6
LINK6               DB  4 DUP(?)
ESRADDRES6          DB  4 DUP(O)
INUSE6              DB  1
CCODE6              DB  ?
```

```
SOCKET6             DW  2007H
IPXWORKSPACE6       DB  4 DUP(?)
DRIVERWORKSPACE6    DB 12 DUP(?)
IMMADDRESS6         DB 6 DUP(0)
FRAGNENTCNT6        DW 2
ADDRESS61           DW  OFFSET CHECKSUM6
                    DW  SEG CHECKSUM6
                    DW  42
ADDRESS62           DW  OFFSET STROKA6
                    DW  SEG STROKA6
SIZ6                DW  534
; пакет и блок ECB для установления связи
;**/////*********************************\\\\\******
;заголовок пакета 2 для установления связи
CHECKSUM2           DB 2 DUP(0)
LEN2                DB 2 DUP(0)
TRANSPORTCONTROL2   DB 0
PACKETTYPE2         DB 5
DESTNETWORK2        DB 00
                    DB 00
                    DB 19H
                    DB 58H
DESTNODE2           DB OOH
                    DB 40H
                    DB 95H
                    DB OEOH
                    DB 9FH
                    DB 67H
DESTSOCKET2         DW 1007H
SOURCENETWORK2      DB 4 DUP(?)
SOURCENODE2         DB 6 DUP(?)
SOURCESOCKET2       DB 2 DUP(?)
;SPX
CONNCONTROL2        DB 0
DATASTREAMTYPE2     DB 0
SOURCECONNID2       DW 0
DESTCONNID2         DW 0
SEQNUMBER2          DW 0
ACKNUMBER2          DW 0
ALLOCNUMBER2        DW 0
;***********************************
;здесь блок ECB 2
LINK                DB   4 DUP(0)
ESRADDRES           DB   4 DUP(0)
```

```
INUSE              DB  0
CCODE              DB  0
SOCKET             DW  2007H
IPXWORKSPACE       DB  4 DUP(?)
DRIVERWORKSPACE    DB 12 DUP(?)
IMMADDRESS         DB 00H
                   DB 40H
                   DB 95H
                   DB OEOH
                   DB 9FH
                   DB 67H
FRAGNENTCNT        DW  1
ADDRESS1           DW  OFFSET CHECKSUM2
                   DW  SEG CHECKSUM2
                   DW  42
;область сообщений
TEXT1  DB  "Сетевые функции отсутствуют.",13,10,'$'
TEXT2  DB  "Сокет не удалось открыть",13,10,'$'
TEXT3  DB  "Канал установлен.",13,10,'$'
TEXT4  DB  "Канал не устанавливается.",13,10,"$"
                   _DX DW ?
DATA ENDS
STAC SEGMENT STACK
      DW 100 DUP(?)
STAC ENDS
CODE SEGMENT
      ASSUME CS:CODE, DS:DATA, SS:STAC, ES:DATA
BEGIN:
      MOV  AX,DATA
      MOV  DS,AX
      CALL WHAT_POINT
      CMP AL,0FFH
      JZ  _OK1
      LEA DX,TEXT1
      MOV AH,9
      INT 21H
      JMP _END
_OK1:
;протокол SPX присутствует
;открыть сокет
      XOR  BX,BX
      MOV  AL, OH
      MOV  DX,2007H
      CALL DWORD PTR CS:NET_POINT
```

```
        CMP   AL,0
        JZ    OK_OPEN
;сокет не удалось открыть
        LEA DX,TEXT2
        MOV AH,9
        INT 21H
        JMP _END
OK_OPEN:
;сокет открыт
        PUSH DS
        POP ES
;пятикратный вызов функции приема пакета  (17H)
        LEA  SI,ES:LINK1
        MOV  BX,17H
        CALL  DWORD  PTR  CS:NET_POINT
        LEA  SI,ES:LINK3
        MOV  BX,17H
        CALL  DWORD  PTR  CS:NET_POINT
        LEA  SI,ES:LINK4
        MOV  BX,17H
        CALL  DWORD  PTR  CS:NET_POINT
        LEA  SI,ES:LINK5
        MOV  BX,17H
        CALL  DWORD  PTR  CS:NET_POINT
        LEA  SI,ES:LINK6
        MOV  BX,17H
        CALL  DWORD  PTR  CS:NET_POINT
;-------------------------------------------
;установить соединение .
        MOV   BX,11H
        MOV  AL,0
        MOV  AH,0
        LEA  SI,ES:LINK
        CALL  DWORD  PTR  CS:NET_POINT
;проверить,  нет  ли  ошибки
        CMP  AL, 0
        JZ    WAI
        LEA   DX,TEXT4
        MOV  AH,9
        INT  21H
        JMP  SHORT  _CLOSE
;ждем установления  канала
WAI:
        MOV   AH,1
        INT   16H
```

```
        JNZ  _CLOSE
        CALL FOR_WAIT
        CMP DS:INUSE,0
        JNZ WAI
        MOV _DX,DX .
;канал установлен
        LEA DX,TEXT3
        MOV AH,9
        INT 21H
_ABORT:
        MOV DX,_DX
        MOV BX,14H
        CALL DWORD PTR CS:NET_POINT
_CLOSE:
        MOV BX,1
        MOV DX,2007H
        CALL DWORD PTR CS:NET_POINT
_END:
        MOV AX,4C00H
        INT 21H
;область процедур
;определение наличия сетевых протоколов и
;определение точки входа в сетевой драйвер
WHAT_POINT  PROC
        MOV AX,7A00H
        INT 2FH
        CMP AL,0FFH
        JNZ NO_IPX_SPX
        MOV CS:NET_POINT,DI
        MOV CS:NET_POINT+2,ES
;проверяем наличие SPX
        XOR AL,AL
        MOV BX,10H
        CALL DWORD PTR CS:NET_POINT
NO_IPX_SPX:
        RETN
WHAT_POINT ENDP
;процедура, вызываемая при ожидании
FOR_WAIT PROC
        PUSH BX
        MOV BX,0AH
        CALL DWORD PTR CS:NET_POINT
        POP BX
        RET
```

```
FOR_WAIT ENDP
PAUSE  PROC
       PUSH  CX
       MOV   CX,20
DAL:
       PUSH  CX
       MOV   CX,0FFFFH
PA:    LOOP  PA
       POP   CX
       LOOP  DAL
       POP   CX
       RET
PAUSE  ENDP
;адрес вызова сетевых процедур
NET_POINT  DW ?
       DW ?
CODE  ENDS
       END  BEGIN
```

Рис. 21.5 Пример двух программ, устанавливающих канал друг с другом по SPX протоколу.

V. Работа с сервером (интерфейс 21-го прерывания).

Предварительные замечания.

Сетевые программы, запускаемые на рабочих станциях, не только поддерживают сетевые протоколы, но и работу сетевых устройств. Осуществляется это посредством перехвата 21-го прерывания. Если запрос, идущий через это прерывание относится к несетевому устройству, то сетевой обработчик передает управление стандартному обработчику. Если же запрос касается сетевого устройства, то сетевой обработчик берет его на себя и передает соответствующие команды на сервер. Сигналы от сервера о результате выполненной операции преобразуются им в стандартный DOS-овский отклик. Таким образм, большинство программ может работать с сетевыми устройствами так же, как с обычными разделами жесткого диска или дискетами.

Все функции 21-го прерывания, которые берет на себя сетевой обработчик, можно разделить на три класса:

1. Стандартные DOS-овские процедуры, но по отношению к каталогам и файлам на сетевых устройствах. Это функции открытия файла, закрытия файла, записи в файл и т.д. Нужно только помнить, что все это делает не операционная система на рабочей станции, а сетевой обработчик и ОС на сервере.

2. Функции работы с сетевыми каталогами и файлами, дополняющими стандартные.

3. Дополнительные функции, расширяющие возможности управления локальной сетью с рабочей станции.

Ниже перечислены функции, относящиеся ко 2-й и 3-й группам, и приведены примеры их использования.

Краткий справочник. Типы объектов.

Тип объекта	Название
0	неизвестный объект
1	пользователь
2	группа пользователей
3	очередь печати
4	файл-сервер
5	прикладной сервер
6	шлюз
7	сервер печати
8	очередь архивирования
9	сервер для архивирования
A	очередь заданий
B	администратор
26H	удаленный мост
до 8000H	зарезервировано
FFFFH	универсальный тип

Объект имеет следующие характеристики.

Идентификатор - число, размером 4 байта, являющееся уникальным для данного файл-сервера.

Имя объекта - строка, не превышающая 47 байт. При поиске объекта в базе объектов допускаются шаблоны, подобные используемым в MS DOS.

Флаг объекта - характеризует время жизни объекта. О - статический объект, 1 - динамический, т.е. удаляется автоматически при исчезновении соответствующего сетевого ресурса.

Байт доступа - определяет права объекта, первые четыре бита - отвечают за чтение, старшие - за запись.

Вот возможные значения уровней доступа:

0 - объект не зарегистрирован.

1 - объект зарегистрирован.

2 - объект подключен к файл-серверу с именем и паролем.

3 - объект имеет права супервизора.

4 - объект имеет права операционной системы.

Флаг свойства объекта показывает, обладает ли объект некоторым свойством: 0 - нет свойств, FFH - имеет по крайней мере одно свойство.

Для каждого свойства имеются: имя свойства - идентифицирует свойство и может содержать до 15 символов, флаги свойства - показывают две вещи - является ли свойство статическим или динамическим и является ли оно единичным или множественным. Это определяется двумя младшими битами:

xO - статическое,
x1 - динамическое
Ox - отдельное
1x - набор свойств.

Имя пользователя пример единичного свойства, список пользователей группы - пример множественного свойства.

Защита свойства - определяет, кто может иметь доступ к свойству для просмотра и изменения (аналогично, как и для объекта)

Флаг значений свойства - как и для объекта.

Значение свойства хранятся в виде 128-байтовых сегментов. Может существовать более чем один сегмент. Каждый сегмент нумеруется, и каждая операция чтения или записи осуществляется над одним целым сегментом.

Перечень функций по группам.

1. Подключение к файл-серверу.

Проверка присутствия сетевой оболочки.
Вход:
AX=OEA01H
ES:DI= указатель на буфер размером 40 байт, в который будет записано текстовое описание среды рабочей станции. Описание состоит из 4 строк:

- название операционной системы,
- версия операционной системы,
- модель компьютера,
- фирма-производитель компьютера.

Выход:
BH=верхний номер версии сетевой ОС или 0, если сетевая оболочка не загружена или ее номер меньше 2.1,
BL=нижний номер версии,
CL=номер изменения.

Создание канала с файл-сервером.
Вход:
AH=F1H
AL= 0 - создать канал с файл-сервером, используя номер в DL.
= 1 - отключить пользователя и удалить канал, заданный в DL.
= 2 - отключить пользователя от файл-сервера, номер канала в DL.
Выход:
AL=0, если операция выполнена успешно или номер ошибки.

Задание серверов.
Вход:
AH=0FH,
AL=0 - установить предпочтительный файл-сервер, номер канала которого задан в регистре DL,
= 1 - определить текущий предпочтительный сервер, номер сервера возвращается в регистре AL,
= 2 - получить в регистре AL номер текущего сервера,
= 4 - установить первичный файл-сервер, номер канала которого задан в регистре DL,
= 5 - получить в регистре AL номер первичного файл-сервера

Отсоединение станции от файл-сервера.

Вход:
AH = OF1H
AL = 1
DL = идентификатор соединения.
Выход:
AL - код завершения
= 0 успешно
= FFH соединения не существует

Информация об объекте.

Функция возвращает информацию об объекте с известным номером.
Вход:
AH = E3H
DS:SI - указатель на входной буфер
ES:DI - указатель на выходной буфер.
Выход:
AL = 0 завершение успешно.
Входной буфер:
DW ? ;длина буфера
DB ? ;type - 16H
DB ? ;номер соединения
Выходной буфер:
DW ? ;длина буфера
DD ? ;номер объекта. Перевернутый формат
DW ? ;тип объекта. Перевернутый формат.
DB 48 dup(?) ;имя объекта
DB 7 dup(?) ;время регистрации
Формат времени регистрации:
1 - год от 0 до 99 (0 =1980)
2 - месяц (1-12)
3 - день (1-31)
4 - час (0-24)
5 - минуты (0 - 59)
6 - секунды (0-59)
7 - день недели, 0-6 (0 - воскресенье)

Получить номер соединения.

Вход:
AH = DCH
Выход:
AL = номер соединения
CL = первая цифра номера соединения
CH = вторая цифра номера соединения.

Получить адрес соединения в сети.

Адрес состоит из номера сети, адреса рабочей станции и сокета, который можно использовать только для работы с сервером.

Вход:
AH = E3H
DS:SI - указатель на входной буфер
ES:DI - указатель на выходной буфер
Входной буфер:
DW ? ;длина буфера
DB ? ;type - 13H
DB ? ;номер соединения
Выходной буфер:
DW ? ;длина буфера
DB 4 dup(?) ;номер сети
DB 6 dup(?) ;номер рабочей станции
DB 2 dup(?) ;сокет

Получить номер соединения объекта.

Возвращает массив номеров соединений, под которыми данный объект зарегистрирован.

Вход:
AH = E3H
DS:SI - указатель на входной буфер
ES:DI - указатель на выходной буфер
Выход:
AL = 0 - успешно.
Входной буфер:
DW ? ;длина буфера
DB ? ;type = 15H
DW ? ;тип объекта (перевернутый формат)
DB ? ;длина имени
DB 48 dup(?) ;имя объекта
Выходной буфер:
DW ? ;длина буфера
DB ? ;число соединений
DB 100 dup(?) ;номера соединений

Регистрация объекта на файловом сервере по умолчанию.

Вход:
AH = E3H
DS:SI - указатель на входной буфер
ES:DI - указатель на выходной буфер
Выход:
AL = 0 успешно
= FFH неверный пароль
Входной буфер:
DW ? ;длина буфера
DB ? ;type - 14H
DW ? ;тип объекта (перевернутый формат)
DB ? ;длина имени объекта

DB 48 dup(?) ;имя объекта (заглавные буквы)
DB ? ;длина пароля
DB 128 dup(?) ;пароль (заглавные буквы и зашифрован)

Выходной буфер:
DW ? ;длина буфера

Завершение сеанса со всеми серверами.

Вход:
AH = D7H

Завершение сеанса с данным сервером.

Вход:
AH = F1H
AL = 2
DL = идентификатор соединения сервера (1-8)

Получение даты и времени на сервере.

Вход:
AH = E7H
DS:DX - адрес буфера, размером 7 байт.

Выход:
Выходной буфер:

Байт 0 год 0 до 99, значения меньше 80 относятся к XXI веку
1 месяц (1-12)
2 день месяца (1-31)
3 час (0-23)
4 минуты (0-59)
5 секунды (0-59)
6 день недели (0-6, 0 - воскресенье)

Установка даты и времени на сервере.

Вход:
AH = E1H
DS:SI - входной буфер,
ES:DI - выходной буфер.

Выход:
AL - 0 или код ошибки.

Входной буфер:
DW ? ;длина буфера
DB ? ;type = 202
DB ? ;год
DB ? ;месяц
DB ? ;день
DB ? ;часы
DB ? ;минуты
DB ? ;секунды

Выходной буфер:
DW ? ;длина буфера

Останов файл-сервера.

Вход:
AH = E3H
DS:SI - входной буфер
ES:DI - выходной
Выход:
AL 0 или код ошибки.
Входной буфер:
DW ? ;длина буфера
DB?;type = 211
DB ? ;FFH - завершить в любом случае
00 - завершить, если нет открытых файлов
Выходной буфер:
DW? ;длина буфера

Запрет и разрешение подключения к серверу.

Вход:
AH = E3H
DS:SI - входной буфер
ES:DI - выходной
Выход:
AL 0 или код ошибки.
Входной буфер:
DW ? ;длина буфера
DB ? ;type = 203 - запретить, 204 - разрешить.
Выходной буфер:
DW ? ;длина буфера

2. Работа с томами и каталогами.

Здесь представлены дополнительные функции (по отношению к функциям MS DOS) работы с каталогами.

Соответствие между именем тома и его номером (имя по номеру тома).

Вход:
AH = E2H
DS:SI - указатель на входной буфер
ES:DI - указатель на выходной буфер
Входной буфер:
DW ? дайна буфера
DB ? ;type - 6
DB ? ;номер тома
Выходной буфер:
DW ? ;длина буфера
DB ? ;длина имени тома
DB 16dup(?) ;имя тома

Если нет соответствия, то длина имени тома будет равна нулю.

Соответствствие между именем тома и его номером (номер по имени).
Вход:
AH = E2H
DS:SI - указатель на входной буфер
ES:DI - указатель на выходной буфер
Входной буфер:
DW ? ;длина буфера
DB ? ;type - 5
DB ? ;длина имени тома
DB 16 dup(?) ;имя тома
Выходной буфер:
DW ? дайна буфера
DB ? ;номер тома.

Информация о томе.
Вход:
AH = E2H
DL = номер тома.
ES:DI - указатель на выходной буфер
Выход:
Al = 0 или код ошибки.
Выходной буфер:
DW ? ;секторов на блок
DW ? ;всего блоков
DW ? ;доступных блоков
DW ? ;всего каталогов
DW ? ;количество каталогов, которые можно создать
db 16 dup(?) ;имя тома
DW ? ;признак съемности тома

Отображение локального диска на сетевой каталог.
Вход:
AH = E2H
DS:SI - указатель на входной буфер
ES:DI - указатель на выходной буфер
Выход:
AL - 0 или код ошибки.
Входной буфер:
DW ? ;длина буфера
DB ? ;type - 18
DB ? ;индекс каталога
DB ? ;отображаемый диск
DB ? ;длина пути к каталогу
DB ? ;путь к каталогу
Выходной буфер:
DW ? ;длина буфера

DB ? ;новый индекс каталога
DB ? ;маска прав доступа

Для временного отображения можно воспользоваться также функцией, но с полем type=19. Для удаления соответствующего отображения, можно воспользоваться той же функцией с таким входным буфером:

DW ? ;длина буфера
DB ? ;type - 20
DB ? ;индекс каталога

Получение адресов таблиц.

Вход:

AH = EFH
AL = 0 - получить адрес таблицы индексов дисковых устройств (32 байта),
1 - получить адрес таблицы флагов дисковых устройств (32 байта),

возможные флаги:

0 - диска нет,
1 - диск постоянно отображен на сетевой,
2 - диск временно отображен на сетевой,
80H - локальный диск,
81H - локальный диск постоянно отображенный на сетевой,
82H - локальный диск временно отображенный на сетевой.

2 - получить адрес таблицы номеров каналов дисковых устройств (32 байта),
3 - получить адрес таблицы номеров каналов,
4 - получить адрес таблицы имен серверов (48 байт).

Выход:

ES:SI - адрес таблицы.

Просмотр подкаталога в заданном подкаталоге.

Вход:

AH = E2H
DS:SI - указатель на входной буфер
ES:DI - указатель на выходной буфер

Входной буфер:

DW ? ;длина пакета
DB ? ;type - 2
DB ? ;индекс каталога вначале 0, затем значение
;из выходного буфера, оба значения имеют перевернутый формат
DW ? ;порядковый номер
DB ? ;длина пути
DB ? ;путь

Выходной буфер:

DW ? ;длина буфера
DB 16 DUP(?) ;имя найденного каталога
DB 2 DUP(?) ;дата создания
DB 2 DUP(?) ;время создания
DB 4 DUP(?) ;идентификатор пользователя, создавшего кат.

DB ? ; маска прав доступа
DB ? ; резерв
DW? ; номер подкаталога в каталоге

Создание каталога.
Вход:
AH = E2H
DS:SI - указатель на входной буфер
ES:DI - указатель на выходной буфер
Выход:
AL - 0 или код ошибки.
Входной буфер:
DW? ;длина буфера
DB ? ;type - 10
DB ? ;индекс каталога
DB ? ;маска прав каталога
DB ? ;длина пути к каталогу
DB ? ;путь к каталогу

Для удаления каталога та же функция со следующим буфером.
Входной буфер:
DW? ;длина буфера
DB ? ;type - 11
DB ? ;индекс каталога
DB ? ;не используется
DB ? ;длина пути к каталогу
DB ? ;путь к каталогу

Для изменения имени каталога та же функция со следующим буфером.
Входной буфер:
DW? ;длина буфера
DB ? ;type - 15
DB ? ;индекс каталога
DB ? ;длина пути к каталогу
DB ? ;путь к каталогу
DB ? ;длина нового пути к каталогу
DB ? ;новый путь к каталогу

Получить права доступа к каталогу.
Вход:
AH = E2H
DS:SI - указатель на входной буфер
ES:DI - указатель на выходной буфер
Выход:
AL - 0 или код ошибки.
Входной буфер:
DW? ;длина буфера

DB ? ;type - 3
DB ? ;индекс каталога
DB ? ;длина пути к каталогу
DB ? ;путь к каталогу

Изменение атрибута каталога.

Вход:
AH = E2H
DS:SI - указатель на входной буфер
ES:DI - указатель на выходной буфер

Выход:
AL - 0 или код ошибки.

Входной буфер:
DW ? ;длина буфера
DB ? ;type - 3
DB ? ;индекс каталога
DB 4 DUP(?) ;новые дата и время
DB 4 dup(?) ;идентификатор владельца
DB ? ;маска прав доступа
DB ? ;длина пути к каталогу
DB ? ;путь к каталогу

Для изменения маски прав доступа используется та же функция со следующим входным буфером.

Входной буфер:
DW ? ;длина буфера
DB ? ;type - 4
DB ? ;индекс каталога
DB ? ;удаляемые права доступа
DB ? добавляемые права доступа
DB ? ;длина пути к каталогу
DB ? ;путь к каталогу

```
CODE   SEGMENT
       ORG  100H
       ASSUME CS:CODE, DS:CODE, SS:CODE, ES:CODE
BEGIN:
       LEA  SI,BUF_IN
       LEA  DI,BUF_OUT
       MOV BYTE PTR N_VOL, OFFH
;вводим длины буферов
       MOV WORD PTR BUF_IN,4     ;длина входного буфера
       MOV WORD PTR BUF_OUT,19  ;длина выходного буфера
CON:
       INC  BYTE  PTR  N_VOL
       MOV  AH,0E2H
```

```
        INT  21H
        CMP  AL,0
        JZ   OK
        LEA  DX,MES_ER
        MOV  AH, 9
        INT  21H
        JMP  SHORT _END
OK:
;проверим длину
        CMP  LEN,0   ;если 0 - список томов кончился
        JZ   _END
;вывод имени тома
        MOV  BL,LEN
        XOR  BH,BH
        MOV  BYTE PTR NAM+[BX],13
        MOV  BYTE PTR NAM+[BX]+1,10
        MOV  BYTE PTR NAM+[BX]+2, '$'
        LEA  DX,NAM
        MOV  AH,9
        INT  21H
        JMP  SHORT CON
_END:
        RET
;блок данных
BUF_IN   DW ?
        DB 6
N_VOL    DB 0
BUF_OUT  DW ?
LEN      DB ?
NAM      DB 16 DUP(?)
MES_ER   DB 'Произошла ошибка!',13,10,'$'
CODE ENDS
        END BEGIN
```

Рис. 21.6. Вывод имен томов сервера по умолчанию.

```
CODE SEGMENT
        ORG 100H
        ASSUME CS:CODE, DS:CODE, ES:CODE, SS:CODE
BEGIN:
        MOV BUF_IN,17  ;длина входного буфера
        MOV BUF_OUT, 4  ;длина выходного буфера
        LEA SI,BUF_IN
        LEA DI,BUF_OUT
```

```
        MOV  AH,0E2H
        INT  21H
        CMP  AL,0
        JZ  _END
;сообщение об ошибке
        LEA  DX,MES
        MOV  AH,9
        INT  21H
_END:
        RET
;область данных
;входной буфер
BUF_IN     DW ?    ;17
        DB  18
        DB  0    ;индекс не заполняем
DISK       DB  'G' ;имя устройства
        DB  11
        DB  'SYS:PUBLIC',0 ;каталог для отображения
;выходной буфер
BUF_OUT   DW ?    ;4
IND_NEW   DB ?
        DB ?
MES      DB  'Произошла ошибка.', 13,10, '$'
CODE  ENDS
        END  BEGIN
```

Рис. 21.7. Простая программа, отображающая устройство G: на сетевой каталог SYS:PUBLIC.

3. Работа с файлами.

Как уже было сказано, сетевая оболочка перехватывает 21-е прерывание и контролирует все обращения к сетевому диску, преобразуя соответствующие команды в запросы серверу. Здесь, естественно, приводятся только дополнительные файловые функции, предоставляемые сетевой оболочкой.

Поиск файла в каталоге.

Вход:

AH = E3H

DS:SI - указатель на входной буфер

ES:DI - указатель на выходной буфер

Выход:

AL = 0 или код ошибки.

Входной буфер:

DW? ;длина буфера

DB ? ;type = 15

DW ? ;вначале FFFFH, а затем декремент из аналогичного поля выходного буфера
DB ? ;индекс устройства
DB ? ;тип файла
DB ? ;длина пути
DB ? ;путь

Выходной буфер:
DW ? ;длина пакета
DW ? ;номер для входного буфера
DB 15 dup(?) ;имя файла
DB ? ;атрибут файла
DB ? ;расширенный атрибут файла
DB 4 dup(?) ;длина файла
DB 2 dup(?) дата создания
DB 2 dup(?) дата последнего доступа
DB 4 dup(?) дата и время обновления
DB 4 dup(?) дата и время выгрузки
DB 60 dup(?) ;резерв

Атрибуты для поиска:
О - обычные файлы,
2 - обычные и скрытые файлы,
4 - обычные и системные файлы,
6 - обычные, скрытые и системные.

Изменение атрибута файла и другой информации.

Вход:
AH = E3H
DS:SI - указатель на входной буфер
ES:DI - указатель на выходной буфер

Выход:
AL = 0 или код ошибки.

Входной буфер:
DW? ;длина буфера
DB ? ;type = 16
DB ? ;атрибут файла
DB ? ;расширенный атрибут файла
DB 4 dup(?) ;резерв
DB 2 dup(?) дата создания
DB 2 dup(?) дата последнего доступа
DB 4 dup(?) дата и время обновления
DB 4 dup(?) дата и время выгрузки
DB 60 dup(?) ;резерв
DB ? ;индекс каталога
DB ? ;тип файла для поиска
DB ? ;длина пути
DB ? ;путь

Получение и изменение расширенных атрибутов.

Вход:

AH = B6H

AL = 0 - получить атрибут

1 - изменить атрибут.

DS:DX - адрес пути к файлу

CL - новое значение атрибута

Выход:

AL - 0 или код ошибки

CL - значение расширенного атрибута.

Копирование файлов.

Вход:

AH = F3H

ES:DI - адрес входного буфера

Выход:

AL - 0 или код ошибки

DX:CX - количество скопированных байт.

Входной буфер:

DW ? дескриптор входного файла

DW ? дескриптор выходного файла

DB 4 dup(?) ;смещение входного файла

DB 4 dup(?) ;смещение выходного файла

DB 4 dup(?) ;сколько байт копировать

Дескрипторы файлов получаются при помощи стандартных функций DOS. Если копируется весь файл, то смещения равны нулю.

```
CODE  SEGMENT
        ORG 100H
        ASSUME CS:CODE, DS:CODE, SS:CODE, ES:CODE
BEGIN:
        MOV  AH,0E3H
        LEA  SI,BUF_IN
        LEA  DI,BUF_OUT
        INT  21H
        CMP AL, 0
        JNZ _END  ;выход, если ошибка или список кончился
;вывод имени
        MOV CX,12
        LEA  BX,NAM
        MOV AH, 2
```

```
LOO:
        MOV DL,[BX]
        INT  21H
        INC  BX
        LOOP  LOO
        MOV DL,13
        INT  21H
        MOV DL,10
        INT  21H
/новый  номер
        MOV  DX,NOM_OU
        MOV  NOM_IN,DX
        JMP  BEGIN
_END:
        RET
;область данных
;входной буфер
BUF_IN     DW 23  /длина  входного  буфера
        DB 15
NOM_IN     DW OFFFFH   ;номер файла
        DB 0
        DB 6  /атрибут,  6-все  файлы
        DB 14  ;длина строки без нуля
        DB  'SYS:PUBLIC\*.*',0
;выходной буфер
BUF_OUT    DW 97  /длина  выходного  буфера
NOM_OU     DW ?
NAM        DB 15  DUP(?)
ATR        DB ?
ATR_EX     DB ?
LEN        DB 4  DUP(?)
DAT1       DB 2  DUP(?)
DAT2       DB 2  DUP(?)
DAT3       DB 4  DUP(?)
DAT4       DB 4  DUP(?)
RES        DB 60  DUP(?)
CODE  ENDS
        END  BEGIN
```

*Рис. 21.8. Программа выводит список файлов каталога **SYS:PUBLIC**, используя **сетевые средства.***

4. Передача и прием сообщений.

Определение режима приема сообщений.

Вход:

AH = DEH

DL = 4

Выход:

AL - номер текущего режима (0, 1, 2, 3)

0 - режим устанавливается по умолчанию. По приходу сообщения на рабочую станцию, оно автоматически отображается в низу экрана (текстовый режим).

1 - файл сервер запоминает в буфере приходящие от пользователей сообщения, на рабочей станции отображаются только сообщения, пришедшие от сервера.

2 - сервер игнорирует сообщения от пользователей и запоминает сообщения только от серверов, вывод сообщений на экран не выполняется.

3 - файл сервер запоминает все пришедшие к нему сообщения, вывод на экран не выполняется.

Установка режима приема сообщений.

Вход:

AH = DEH

DL = новое значение режима.

Передача сообщений пользователем.

Вход:

AH = E1H

DS:SI - адрес входного буфера,

ES:DI - адрес выходного буфера.

Выход:

AL - 0 или код ошибки.

Входной буфер:

DW ? ; длина буфера

DB ? ; type = 0

DB ?; количество станций (элементов массива)

DB ? ; массив номеров каналов

DB ?; длина сообщения

DB ? ; сообщение

Выходной буфер:

DW ?; длина буфера

DB ?; количество станций

DB ?; массив байт - результат для каждой станции

Прием сообщений.

Вход:

AH = E1H

DS:SI - адрес входного буфера,
ES:DI - адрес выходного буфера.
Выход:
AL - 0 или код ошибки.
Входной буфер:
DW ? ; длина буфера
DB ? ; type = 1
Выходной буфер:
DW ? ; длина буфера
DB ? ; длина сообщения
DB ? ; сообщение

Посылка строки на сервер.
Вход:
AH = E1H
DS:SI - адрес входного буфера,
Выход:
AL - 0 или код ошибки.
Входной буфер:
DW ? ; длина буфера
DB ? ; type = 9
DB ? ; длина строки
DB ? ; строка

```
CODE SEGMENT
     ASSUME CS:CODE, DS:CODE, ES:CODE
     ORG 100H
BEGIN:
;определяем количество соединений -
;пользователей с именем RUFAT
     MOV AH,0E3H
     LEA SI,BUF_IN
     LEA DI,BUF_OUT
     INT 21H
     CMP AL,0
     JZ  OK
     LEA DX,MES
     MOV AH,9
     INT 21H
     JMP SHORT _END
OK:
```

```
;формируем буфер посылки сообщений
       XOR CH,CH
       MOV CL,N_CON
       LEA SI,N_CON
       INC SI
       LEA DI,MAS
       CLD
LOO:
       LODSB
       STOSB
       LOOP LOO
;осуществить пересылку
       LEA SI,BUF_IN_SEND
       LEA DI,BUF_OUT_SEND
       MOV AH,0E1H
       INT 21H
_END:
       RET
BUF_IN   DW 11
       DB 15H
       DW 0100H     ;тип объекта - пользователь (1)
       DB 5         ;длина имени пользователя
       DB 'RUFAT'   ;имя пользователя
BUF_OUT DW 103
N_CON    DB ?        ;количество регистрации (рабочих станций)
       DB 100 DUP(?)  ;номера соединений
BUF_IN_SEND DW 120
       DB 0
N_SEND    DB 100 ;в действительности посылается N_CON машинам
MAS       DB 100 DUP(?)
       DB 15 ;длина строки
       DB 'Проверка связи!' ;строка для посылки
BUF_OUT_SEND DW 103
       DB 100
       DB 100 DUP(?)
MES       DB 'Ошибка!',13,10,'$'
CODE ENDS
       END BEGIN
```

Рис. 21.9. Пример посылки сообщений всем пользователям, зарегистрированным под именем R UFA T.

```
DATA SEGMENT
;структура для передачи строки на консоль LEN_BUF DW 40H
       DB 09H
LEN    DB ?
MES    DB 60 DUP(?)
;структура для ввода строки
MAX       DB 60
LENS      DB ?
STROKA    DB 61 DUP(?)
DATA ENDS
ST1 SEGMENT STACK
       DB 100 DUP(?)
ST1 ENDS
CODE SEGMENT
       ASSUME CS:CODE, DS:DATA, SS:ST1
BEGIN:
       MOV AX,DATA
       MOV DS,AX
       MOV ES,AX
;ввод строки (-> STROKA)
       MOV AH,0AH
       LEA DX,MAX
       INT 21H
;копирование строки (STROKA -> MES)
       CLD
       LEA SI,STROKA
       LEA DI,MES
       MOV CL,LENS
       MOV LEN,CL
       XOR CH,CH
       REP MOVSB
;передача на консоль (MES -> CONSOLE)
       LEA SI,LEN_BUF
       MOV AH,0E1H  ;сетевая функция E1H
       INT 21H
_END:
       MOV AH,4CH
       INT 21H
CODE ENDS
       END BEGIN
```

Рис. 21.10. Программа передает на консоль вводимую строку.

5. Работа с объектами.

Добавление объекта к множественному свойству.

Вход:

AH = E3H
DS:SI - адрес входного буфера
ES:DI - адрес выходного.

Выход:

AL - 0 или код ошибки.

Входной буфер:

DW ? ;длина буфера
DB ? ;type - 41h
DW ? ;тип объекта (обратный порядок)
DB ? ;длина имени (48)
DB ? ;имя объекта
DB ? ;длина имени свойства (16)
DB ? ;имя свойства
DW ? ;тип добавляемого объекта
DB ? ;длина имени добавляемого объекта (48)
DB ? ;имя добавляемого объекта

Выходной буфер:

DW ? ;длина буфера

Изменяет пароль объекта.

Вход:

AH = E3H
DS:SI - адрес входного буфера
ES:DI - адрес выходного.

Выход:

AL - 0 или код ошибки.

Входной буфер:

DW ? ;длина буфера
DB ? ;type - 40h
DW ? ;тип объекта (обратный порядок)
DB ? ;длина имени (48)
DB ? ;имя объекта
DB ? ;длина старого пароля (127)
DB ? ;старый пароль
DB ? ;длина нового пароля (127)
DB ? ;новый пароль

Выходной буфер:

DW ? ;длина буфера

Изменение защиты объекта.

Вход:

AH = E3H

DS:SI - адрес входного буфера

ES:DI - адрес выходного.

Выход:

AL - 0 или код ошибки.

Входной буфер:

DW ? ;длина буфера

DB ? ;type - 38h

DB ? ;новый байт защиты

DW ? ;тип объекта (обратный порядок)

DB ? ;длина имени объекта

DB ? ;имя объекта

Выходной буфер:

DW ? ;длина буфера

Изменяет защиту конкретного свойства объекта.

Вход:

AH = E3H

DS:SI - адрес входного буфера

ES:DI - адрес выходного.

Выход:

AL - 0 или код ошибки.

Входной буфер:

DW ? ;длина буфера

DB ? ;type - 3Bh

DW ? ;тип объекта (обратный порядок)

DB ? ;длина имени объекта

DB ? ;имя объекта

DB ? ;байт защиты

DB ? ;длина имени свойства (6)

DB ? ;имя свойства

Выходной буфер:

DW ? ;длина буфера

Закрытие базы объектов.

Вход:

AH = E3H

DS:SI - адрес входного буфера

ES:DI - адрес выходного.

Выход:

AL - 0 или код ошибки.

Входной буфер:
DW ? ;длина буфера
DB ? ;type - 44h
Выходной буфер:
DW ? ;длина буфера

Создание объекта с данными характеристиками.

Вход:
AH = E3H
DS:SI - адрес входного буфера
ES:DI - адрес выходного.
Выход:
AL - 0 или код ошибки.
Входной буфер:
DW ? ;длина буфера
DB ? ;type - 32h
DB ? ;флаг объекта
DB ? ;байт защиты объекта
DW ? ;тип объекта (обратный порядок)
DB ? ;длина имени объекта
DB ? ;имя объекта
Выходной буфер:
DW ? ;длина буфера

Добавление свойства к объекту.

Вход:
AH = E3H
DS:SI - адрес входного буфера
ES:DI - адрес выходного.
Выход:
AL - 0 или код ошибки.
Входной буфер:
DW ? ;длина буфера
DB ? ;type - 39h
DW ? ;тип объекта (обратный порядок)
DB ? ;длина имени
DB ? ;имя объекта
DB ? ;новый флаг
DB ? ;новый байт защиты
DB ? ;длина имени свойства (16)
DB ? ;имя свойства
Выходной буфер:
DW ? ;длина буфера

Удаление объекта.
Вход:
AH = E3H
DS:SI - адрес входного буфера
ES:DI - адрес выходного.
Выход:
AL - 0 или код ошибки.
Входной буфер:
DW ? ;длина буфера
DB ? ;type - 33h
DW ? ;тип объекта (обратный порядок)
DB ? ;длина имени объекта
DB ? ;имя объекта
Выходной буфер:
DW ? ;длина буфера

Удаляет идентификатор объекта из набора свойств.
Вход:
AH = E3H
DS:SI - адрес входного буфера
ES:DI - адрес выходного.
Выход:
AL - 0 или код ошибки.
Входной буфер:
DW ? ;длина буфера
DB ? ;type - 42h
DW ? ;тип объекта (обратный порядок)
DB ? ;длина имени объекта
DB ? ;имя объекта
DB ? ;длина строки свойств (16)
DB ? ;строка свойств
DW ? ;идентификатор удаляемого объекта (обратный порядок)
DB ? ;длина имени удаляемого объекта
DB ? ;имя удаляемого объекта
Выходной буфер:
DW ? ;длина буфера

Удаляет одно или более свойств объекта.
Вход:
AH = E3H
DS:SI - адрес входного буфера
ES:DI - адрес выходного.
Выход:
AL - 0 или код ошибки.

Входной буфер:
DW ? ;длина буфера
DB ? ;type - 3Ah
DW ? ;тип объекта
DB ? ;длина имени
DB ? ;имя объекта
DB ? ;длина строки свойств (16)
DB ? ;строка свойств
Выходной буфер:
DW ? ;длина буфера

Возвращает уровень доступа рабочей станции к базе объектов.
Вход:
AH = E3H
DS:SI - адрес входного буфера
ES:DI - адрес выходного.
Выход:
AL - 0 или код ошибки.
Входной буфер:
DW ? ;длина буфера
DB ? ;type - 46h
Выходной буфер:
DW ? ;длина буфера
DB ? ;байт доступа
DD ? ;идентификатор объекта для данного пользователя

Возвращает уникальный для объекта идентификационный номер.
Вход:
AH = E3H
DS:SI - адрес входного буфера
ES:DI - адрес выходного.
Выход:
AL - 0 или код ошибки.
Входной буфер:
DW ? ;длина буфера
DB ? ;type - 35h
DW ? ;тип объекта (обратный порядок)
DB ? ;длина имени объекта
DB ? ;имя объекта
Выходной буфер:
DW ? ;длина буфера

Возвращает имя и тип объекта.
Вход:
AH = E3H
DS:SI - адрес входного буфера
ES:DI - адрес выходного.

Выход:
AL - 0 или код ошибки.
Входной буфер:
DW ? ;длина буфера
DB ? ;type - 36h
DD ? ;идентификатор объекта (обратный порядок)
Выходной буфер:
DW ? ;длина буфера
DD ? ;идентификатор объекта (обратный порядок)
DW ? ;тип объекта (обратный порядок)
DB ? ;имя объекта

Показывает, входит ли единичный объект в некоторое множественное свойство другого объекта.

Вход:
AH = E3H
DS:SI - адрес входного буфера
ES:DI - адрес выходного.
Выход:
AL - 0 или код ошибки.
Входной буфер:
DW ? ;длина буфера
DB ? ;type - 43h
DW ? ;тип объекта
DB ? ;длина имени
DB ? ;имя объекта
DB ? ;длина имени свойства
DB ? ;имя свойства
DW ? ;идентификатор объекта (обратный порядок)
DB ? ;длина имени объекта
DB ? ;имя объекта
Выходной буфер:
DW ? ;длина буфера

Повторно открывает базу объектов.

Вход:
AH = E3H
DS:SI - адрес входного буфера
ES:DI - адрес выходного.
Выход:
AL - 0 или код ошибки.
Входной буфер:
DW ? ;длина буфера
DB ? ;type - 45h
Выходной буфер:
DW ? ;длина буфера

Возвращает значение указанного свойства данного объекта в виде 128-байтового сегмента.

Вход:
- AH = E3H
- DS:SI - адрес входного буфера
- ES:DI - адрес выходного.

Выход:
- AL - 0 или код ошибки.

Входной буфер:
```
DW ? ;длина буфера
DB ? ;type - 3Dh
DB ? ;тип объекта
DB ? ;длина имени объекта
DB ? ;имя объекта
DB ? ;номер сегмента (вначале 1)
DB ? ;длина строки свойств
DB ? ;строка свойств
```
Выходной буфер:
```
DW ? ;длина буфера
DB ? ;строка (сегмент) свойства
DB ? ;номер следующего сегмента
DB ? ;байт флагов свойств
```

Переименовывает объект.

Вход:
- AH = E3H
- DS:SI - адрес входного буфера
- ES:DI - адрес выходного.

Выход:
- AL - 0 или код ошибки.

Входной буфер:
```
DW ? ;длина буфера
DB ? ;type - 34h
DB ? ;тип объекта
DB ? ;длина старого имени
DB ? ;старое имя
DB ? ;длина нового имени
DB ? ;новое имя
```
Выходной буфер:
```
DW ? ;длина буфера
```

Ищет объект в базе объектов и возвращает о нем информацию.

Вход:
- AH = E3H
- DS:SI - адрес входного буфера
- ES:DI - адрес выходного.

Выход:
AL - 0 или код ошибки.
Входной буфер:
DW ? ;длина буфера
DB ? ;type - 37h
DW ? ;тип объекта
DD ? ;идентификатор объекта (вначале FFFFFFFFh)
DW ? ;тип объекта
DB ? ;длина имени
DB ? ;имя объекта
Выходной буфер:
DW ? ;длина буфера
DD ? ;идентификатор объекта
DW ? ;тип объекта
DB ? ;имя объекта (48)
DB ? ;флаг
DB ? ;байт доступа
DB ? ;есть свойства

Вначале идентификатор содержит значение FFFFFFFFh, а затем берется из выходного буфера до тех пор, пока в AL не будет FCh.

Возвращает любое свойство заданного объекта.
Вход:
AH = E3H
DS:SI - адрес входного буфера
ES:DI - адрес выходного.
Выход:
AL - 0 или код ошибки.
Входной буфер:
DW ? ;длина буфера
DB ? ;type - 3Ch
DW ? ;тип объекта
DB ? ;длина имени объекта
DB ? ;имя объекта
DD ? ;номер (вначале FFFFFFFFH)
DB ? ;длина строки свойств
DB ? ;строка свойств
Выходной буфер:
DW ? ;длина буфера
DB ? ;строка свойств
DB ? ;флаги свойств
DB ? ;байт доступа
DD ? ;последующий номер
DB ? ;байт свойств
DB ? ;еще свойства

После сравнения пароля, помещенного в поле входного буфера, с паролем, хранимым в базе объектов для данного объекта, возвращает код завершения, показывающий, являются ли пароли эквивалентными.

Вход:
  AH = E3H
  DS:SI - адрес входного буфера
  ES:DI - адрес выходного.
Выход:
  AL - 0 или код ошибки.
Входной буфер:
  DW ? ;длина буфера
  DB ? ;type - 3Fh
  DW? ;тип объекта
  DB ? ;длина имени объекта
  DB ? ;имя объекта
  DB ? ;длина пароля (128)
  DB ? ;пароль
Выходной буфер:
  DW? ;длина буфера

Описывает значение данного свойства данного объекта как 128-байтный объект.

Вход:
  AH = E3H
  DS:SI - адрес входного буфера
  ES:DI - адрес выходного.
Выход:
  AL - 0 или код ошибки.
Входной буфер:
  DW? ;длина буфера
  DB ? ;type - 3Fh
  DW? ;тип объекта
  DB ? ;длина имени объекта
  DB ? ;имя объекта
  DB ? ;номер сегмента
  DB ? ;оставшиеся сегменты
  DB ? ;длина строки свойств объекта
  DB ? ;строка свойств объекта
Выходной буфер:
  DW? ;длина буфера

```
CODE  SEGMENT
        ASSUME CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG 100H
```

```
BEGIN:
        MOV AH,0E3H
        LEA SI,BUF_IN
        LEA DI,BUF_OUT
        INT 21H
        CMP AL,0
        JNZ _END
        MOV AL,ACCES
        MOV DL,AL
        MOV CL,4
        SHR AL,CL
        AND DL,00001111B
        ADD DL,48
        MOV AH,2
        PUSH AX
        INT 21H   ;доступ по чтению
        POP AX
        MOV DL,AL
        ADD DL,48
        INT 21H   ;доступ по записи
_END:
        RET
BUF_IN  DW 3
        DB 46H
BUF_OUT DW 7
ACCES   DB ?
        DD ?
CODE ENDS
        END BEGIN
```

Рис. 21.11. Программа определяет уровни доступа рабочей станции к базе объектов.

```
CODE SEGMENT
        ASSUME CS:CODE, DS:CODE, ES:CODE, SS:CODE
        ORG 100H
BEGIN:
        MOV AH,0E3H
        LEA SI,BUF_IN
        LEA DI,BUF_OUT
        INT 21H
        CMP AL,0
        JZ  OK1
        CMP AL,0FCH
        JZ  _END   ;конец поиска
```

```
;• звуковой сигнал-признак ошибки
      MOV DL,7
      MOV AH,2
      INT 21H
      JMP _END
OK1:
      LEA BX,NAME2
      MOV AH,2
;выводим имя
LOO:
      MOV DL, [BX]
      CMP DL,0
      JZ  LOO1
      INT 21H
      INC BX
      JMP SHORT LOO
LOO1:
      MOV  DL, 13
      INT  21H
      MOV  DL,10
      INT  21H
;готовим ID1 для следующего вызова
      MOV  AX,WORD PTR ID2
      MOV  BX,WORD PTR ID2+2
      MOV  WORD PTR ID1,AX
      MOV  WORD PTR ID1+2,BX
      JMP  BEGIN
_END:
      RET
BUF_IN  DW 57
      DB 37H
ID1     DD OFFFFFFFFH
      DW 100H
LEN1    DB 47
NAME1   DB '*'
      DB 46 DUP(0)
BUF_OUT DW 59
ID2     DD ?
      DW ?
NAME2   DB 48 DUP(?)
FLAG    DB ?
SEC     DB ?
      DB ?
CODE ENDS
      END BEGIN
```

Рис. 21.12. Вывод списка всех пользователей, зарегистрированных на данном сервере.

6. Сервис синхронизации.

Данный набор функций позволяет упорядочить коллективный доступ к файлам, расположенным на сервере. Суть работы этих функций заключается в том, что можно блокировать доступ (вообще или только на изменение) к отдельным областям файла или к целым файлам. Например, при просмотре базы данных только один пользователь может иметь право корректировать определенные записи и т.п.

Разблокирует указанный файл и удаляет его из таблицы регистрации.

Вход:

AH = EDH

DS:DX - на входной буфер

Выход:

AL - 0 или код ошибки

Входной буфер:

BUF_IN db 255 dup(?) ;полный путь к файлу

Разблокирует все файлы в таблице регистрации рабочей станции и удаляет их из таблицы регистрации.

Вход:

AH = CFH

Разблокирует указанную физическую запись и удаляет ее из таблицы регистрации.

Вход:

AH = BEH

BX - дескриптор файла

CX:DX - указатель на запись

SI:DI - длина записи

Выход:

AL - 0 или код ошибки

Разблокирует все физические записи и удаляет их из таблицы регистрации.

Вход:

AH = C4H

Возвращает флаг текущего режима блокировки.

Вход:

AH = C6H

AL = 2

Выход:

AL - флаг

Флаг определяет, будет ли система распознавать значение тайм-аута для определенных вызовов.

Функция пытается блокировать все файлы из таблицы регистрации.

Вход:

AH=CBH

BP - тайм-аут в 1/18 с.

Выход:
AL - 0 - успешно
FEH - истек тайм-аут
FFH - ошибка.

Пытается блокировать все физические записи в таблице регистрации.
Вход:
AH = C2H
AL - директива блокировки
0 - блокировка записей в исключительном режиме,
1 - блокировка Read-Only.
BP - тайм-аут в 1/18 с.
Выход:
AL - 0 - успешно
FEH - истек тайм-аут
FFH - ошибка.

Помещает файл в таблицу регистрации.
Вход:
AH = EBH
AL - директива блокировки
0 - зарегистрировать файл
1 - зарегистрировать и заблокировать файл.
BP - тайм-аут в 1/18 с.(при AL=1)
DS:DX - указатель на путь к файлу.
Выход:
AL - 0 - успешно
96H - сервер выгружен
FEH - истек тайм-аут
FFH - ошибка.

Регистрирует физическую запись в таблице регистрации.
Вход:
AH = BCH
AL - директива
0 - зарегистрировать запись
1 - зарегистрировать и заблокировать запись в исключительном режиме
3 - зарегистрировать и заблокировать запись в режиме Read-Only
BX - дескриптор файла
BP - тайм-аут
CX:DX - позиция записи
SI:DI - длина записи
Выход:
AL - 0 - успешно
96H - сервер выгружен
FEH - истек тайм-аут
FFH - ошибка.

Разблокировать файл без удаления его из таблицы.

Вход:

AH - ECH

DS:DX - указатель на путь к файлу

Выход:

AL - 0 или код ошибки.

Разблокирует все файлы без удаления их таблицы.

Вход:

AH = CDH

Разблокирует физическую запись, не удаляя из таблицы.

Вход:

AH = BDH

BX - дескриптор

CX:DX - позиция записи

SI:DI - длина записи

Выход:

AL - 0 или код ошибки.

Разблокирует все физические записи, не удаляя из таблицы.

Вход:

AH = C3H

Устанавливает флаг режима блокировки.

Вход:

AH = C6H

AL - режим

```
CODE SEGMENT
      ASSUME CS:CODE, DS:CODE, ES:CODE, SS:CODE
      ORG 100H
BEGIN:
;поместить файл в таблицу регистрации
      MOV AH,0EBH
      MOV AL,0
      LEA DX,PATH
      INT 21H
      CMP AL,0
      JNZ _END        ;если ошибка - выйти
;заблокировать файлы из таблицы регистрации
      MOV AH,0CBH
      MOV BP,10
      INT 21H
      CMP AL,0
      JNZ _END        ;если ошибка - выйти
```

```
;открываем файл
        MOV AH,3DH
        MOV AL,2
        LEA  DX,PATH
        INT  21H
        JC   _END
        MOV BX,AX
;ждем нажатия клавиши
;теперь с другой рабочей станции
;можете попытаться просмотреть заблокированный файл
        MOV AH,0
        INT 16H
;закрываем файл
        MOV AH,3EH
        INT 21H
;разблокировать файл
        MOV AH,0EDH
        LEA DX,PATH
        INT 21H
_END:
        RET
PATH   DB  'USER:SOFT\ME\STATUS.ME',0
CODE  ENDS
        END BEGIN
```

Рис. 21.13. Пример блокирования файла на сетевом устройстве.

7. Обслуживание транзакций.

В сети Novel существует механизм автоматического восстановления файлов при сбоях - **TTS** - Transaction Tracking Service. Чтобы для файла работал такой механизм, он должен быть помечен **как транзакционный.**

TTS осуществляет "явное" и "неявное" отслеживание транзакций. Транзакцией называется совокупность следующих действий:

- чтение данных,
- обработка данных,
- запись данных.

"Неявное" означает, что серверная ОС запоминает последовательность измененных **транзакционных** файлов без возможности программного вмешательства в этот процесс. "Явное" дает приложениям определенные возможности управления **транзакционным** процессом.

Позволяет вручную прекратить транзакцию, заставляя файлы, вовлеченные в текущую транзакцию, восстанавливаться.

Вход:
AH = C7H
AL = 3
Выход:
AL = 0 или код ошибки.

Служит для явного указания начала транзакции.

Вход:
AH = C7H
AL = 0
Выход:
AL = 0 или код ошибки.

Дает команду завершения явной транзакции.

Вход:
AH = C7H
AL = 1
Выход:
Флаг переноса = О
AL = 0 успешно
FDH отслеживание транзакций запрещено
FEH транзакция закончена
CX:DX - справочный номер транзакции
Флаг переноса = 1
AL = FFH - нет активной явной транзакции

Дает информацию о пороге прикладного уровня логических или физических записей.

Вход:
AH = C7H
AL = 5
Выход:
AL = 0 или код ошибки
CL - порог блокирования логических записей (0-255)
CH - порог блокирования физических записей.

Дает информацию о пороге прикладного уровня логических или физических записей на уровне рабочих станций.

Вход:
AH = C7H
AL = 7
Выход:
AL = 0 или код ошибки
CL - порог блокирования логических записей (0-255)
CH - порог блокирования физических записей.

Показывает, является ли система управления транзакциями доступной и активной на файловом сервере по умолчанию.

Вход:
AH = C7H
AL = 2
Выход:
AL = 0 - недоступна
1 - доступна
FDH - в текущий момент запрещена.

Устанавливает пороги прикладного уровня логических или физических записей.

Вход:
AH = C7H
AL = 6
CL - порог блокирования логических записей (0-255)
CH - порог блокирования физических записей.
Выход:
AL = 0 или код ошибки

Устанавливает пороги прикладного уровня логических или физических записей на уровне рабочей станции.

Вход:
AH = C7H
AL = 8
CL - порог блокирования логических записей (0-255)
CH - порог блокирования физических записей.
Выход:
AL = 0 или код ошибки

Показывает, завершена ли полностью операция для указанной транзакции.

Вход:
AH = C7H
AL = 4
CX:DX - справочный номер транзакции.
Выход:
AL = 0 или код ошибки

8. Сервис рабочей станции.

Вызывает выполнение завершающих операций как на рабочей станции, так и на файловом сервере, освобождая блокированные файлы и записи, закрывая сетевые и локальные файлы, сбрасывая режимы ошибки и блокирования.

Вход:
AH = D6H
BX = 0 - только для текущего процесса
FFFFH - для всех процессов на рабочей станции.

Данная функция с BX=0 вызывается автоматически по выходу из программы в DOS.

Возвращает информацию об окружении рабочей станции.
Вход:
AH = EAH
AL = 1
BX = 0
ES:DI - указатель на выходной буфер.
Выход:
AH - операционная система рабочей станции (0 - PC DOS)
AL - код модификации
BH - старшая часть номера версии сетевой оболочки
BL - младшая часть номера версии сетевой оболочки
CL - номер редакции сетевой оболочки
ES:SI - указатель на выходной буфер.
Выходной буфер:
Состоит из 40 байт. Строка представляет собой конкатенацию четырех строк с нулевым окончанием. Первая строка содержит имя операционной системы. Вторая строка содержит версию операционной системы. Третья строка содержит идентификацию типа компьютера. Четвертая строка содержит более общий идентификатор типа компьютера.

Включает или выключает автоматическое выполнение завершающих операций по окончанию прикладной программы.
Вход:
AH = BBH
AL = 0 - запрещено
1 - разрешено.
Выход:
AL = 0 или код ошибки

Сообщает, как Вы хотите получать сообщения о критических ошибках.
Вход:
AH = DDH
DL = 0 - спрашивать Abort, Retry, Fail?
1 - не выполнять прерывание 24H, возвращать ошибку NetWare в программу
2 - не выполнять прерывание 24H, возвращать ошибку DOS в программу.
Выход:
AL - предыдущую установку режима ошибки.

9. Сервис печати.

Обслуживает сетевые принтеры и очереди к ним.

Структура PRINT_CONTROL_DATA.

```
status db ?
print_flags db ?
tab_size db ?
server_printer db ?
```

```
number_copies db ?
form_type db ?
reservl db ?
banner_text db 13 dup(?)
reserv2 db ?
local_lpt_device db ?
flush_timeout_count dw ?
flush_on_close db ?
maximum_lines dw ?
maximum_chars dw ?
form_name db 13 dup(?)
lpt_flag db ?
file_flag db ?
timeout_flag db ?
setup_string_ptr dd ?
reset_string_ptr dd ?
connect_id_queue_print_job db ?
in_progres db ?
print_queue_flag db ?
print_job_valid db ?
print_queue_id dd ?
print_job_number dw ?
```

Прекращает перехват потока данных на печать по умолчанию. Задание на печать уничтожается.

Вход:

AH=DFH

DL=2

Выход:

AL - 0 или код ошибки.

Прекращает перехват потока данных на определенную печать. Задание на печать уничтожается.

Вход:

AH=DFH

DL=6

DH - LPT (0 - LPT1, 1 - LPT2, 2 - LPT3)

Выход:

AL - 0 или код ошибки.

Освобождает задание на печать принтера по умолчанию и заканчивает перехват для печати по умолчанию.

Вход:

AH=DFH

DL=1

Выход:

AL - 0 или код ошибки.

Освобождает задание на печать для определенного принтера и заканчивает перехват.

Вход:
AH=DFH
DL=5
DH - LPT (0 - LPT1, 1 - LPT2, 2 - LPT3)

Выход:
AL - 0 или код ошибки.

Освобождает задание на печать принтера по умолчанию, но продолжает перехват данных.

Вход:
AH=DFH
DL=3

Выход:
AL - 0 или код ошибки.

Освобождает задание на печать для определенного принтера, но продолжает перехват данных.

Вход:
AH=DFH
DL=7
DH - LPT (0 - LPT1, 1 - LPT2, 2 - LPT3)

Выход:
AL - 0 или код ошибки.

Возвращает имя пользователя, которое печатается на первой странице.

Вход:
AH = B8H
AL=8
ES:BX - длинный указатель на буфер ответа (12 байт)

Выход:
AL - 0 или код ошибки.

Возвращает заполненную структуру PRINT_CONTROL_DATA.

Вход:
AH = B8H
AL = 0
CX - длина буфера ответа
ES:BX - длинный указатель на буфер ответа

Выход:
AL - 0 или код ошибки.

Возвращает номер, соответствующий устройству печати по умолчанию, в настоящий момент назначенному для перехвата.

Вход:
AH = B8H
AL = 4

Выход:
DH - устройство LPT.

Возвращает текущий статус заданного сетевого принтера.
Вход:
AH = E0H
DS:SI - входной буфер
ES:DI - выходной буфер
Выход:
AL - 0 или код ошибки.
Входной буфер:
DW ? ;длина буфера
DB ? ;type = 6
DB ? ;номер принтера (0 - 4)
Выходной буфер:
DW ? ;длина буфера
DB ? ;0 - активный, FFH - остановленный
DB ? ;1 - выключен
DB ? ;(0-255) - form_type
DB ? ;номер принтера

Возвращает заполненную структуру PRINT_CONTROL_DATA, определенную ранее, для заданного устройства печати.
Вход:
AH = B8H
AL = 2
ES:BX - длинный указатель на буфер ответа
DH - устройство LPT
Выход:
AL - 0 или код ошибки.

Устанавливает имя пользователя, которое печатается на первой странице.
Вход:
AH = B8H
AL = 9
ES:BX - длинный указатель на буфер с именем (12 байт)
DH - устройство LPT
Выход:
AL - 0 или код ошибки.

Осуществляет захват следующего принтера для указанной очереди.
Вход:
AH = B8H
AL = 6
BX:CX - идентификатор очереди в базе объектов
DH - устройство LPT
Выход:
AL - 0 или код ошибки.

Можно изменить некоторые поля структуры PRINT_CONTROL_DATA для устройства печати по умолчанию. Изменениям подвергаются поля до FORM_NAME, исключая поле STATUS.

Вход:
AH = B8H
AL = 1
CX - длина входного буфера (часть PRINT_CONTROL_DATAдо 42 байт)
ES:BX - указатель на входной буфер.

Выход:
AL - 0 или код ошибки.

Определяет локальный принтер по умолчанию. Определение действует до следующего вызова этой функции или до перезагрузки рабочей станции.

Вход:
AH = B8H
AL = 5
DH - устройство LPT

Выход:
AL - 0 или код ошибки.

Можно изменить некоторые поля структуры PRINT_CONTROL_DATA для указанного устройства печати.

Вход:
AH = B8H
AL = 3
CX - длина входного буфера (часть PRINT_CONTROL_DATAдо 42 байт)
ES:BX - указатель на входной буфер.
DH - устройство LPT.

Выход:
AL - 0 или код ошибки.

Создает файл на диске и направляет последующие захваченные потоки данных печати в этот файл.

Вход:
AH = E0H
AL = 9
DS:SI - входной буфер
ES:DI - выходной буфер

Выход:
AL - 0 или код ошибки.

Входной буфер:
DW ? ;длина буфера
DB ? ;type = 9
DB ? ;0
DB ? ;длина пути к файлу
DB ? ;полный путь к файлу

Выходной буфер:
DW ? ;длина буфера

Функция начинает захват данных печати на устройство печати по умолчанию.

Вход:

AH = DFH

AL = 0

Выход:

AL - 0 или код ошибки.

Начинает захват данных печати на указанное устройство печати.

Вход:

AH = DFH

AL = 4

DH = устройство LPT.

Выход:

AL - 0 или код ошибки.

10. Семафоры.

Семафорный сервис - еще один способ синхронизации работы программ на разных рабочих станциях. Этот сервис поддерживается сетевой операционной системой[60] на сервере. Семафор имеет следующие характеристики:

- имя семафора - задается при открытии, с данным семафором разные программы могут связать некоторый критический ресурс, например, доступ к определенным записям файла и т.п.
- индекс семафора - возвращается при открытии, в дальнейшем через него осуществляется доступ к данному семафору;
- значение семафора - число в диапазоне -127 - +127, которое также связывается с данным семафором;
- счетчик процессов - увеличивается на 1, когда очередная программа открывает данный семафор, и уменьшается, когда закрывает, когда счетчик принимает значение 0, семафор ликвидируется.

Кратко смысл работы с семафорами заключается в следующем: перед использованием критического ресурса, с которым связан семафор, программа должна уменьшить значение семафора на 1. Если значение семафора больше или равно 0, то программа может использовать ресурс, если же нет, то программа ожидает изменения значения, указанное ей время (время ожидания).

Открыть семафор.

Вход:

AH = C5H

AL = 0

DS:DX - адрес имени семафора (не более 127 символов с 0 на конце)

CL - начальное значение семафора (обычно 1).

Выход:

AL - 0 или код ошибки.

---

[60] Правильнее сказать, что семафор является неотъемлемой частью любой многозадачной операционной системы.

Определить состояние семафора.
Вход:
AH = C5H
AL = 1
CX,DX - индекс семафора
Выход:
AL - 0 или код ошибки
CX - значение семафора
DL - счетчик использования семафора.

Уменьшить значение семафора.
Вход:
AH = C5H
AL = 2
CX,DX - индекс семафора
BP - время ожидания
Выход:
AL - 0 или код ошибки.

Увеличить значение семафора.
Вход:
AH = C5H
AL = 3
CX,DX - индекс семафора
Выход:
AL - 0 или код ошибки.

Закрыть семафор.
Вход:
AH = C5H
AL = 4
CX,DX - индекс семафора
Выход:
AL - 0 или код ошибки.

11. Коды ошибок.

01H - неправильный код функции DOS, сервер используется
02H - файл не найден
03H - путь не найден
04H - много открытых файлов
05H - доступ запрещен
06H - неправильный индекс файла
07H - блок памяти уничтожен
08H - мало памяти
09H - неправильный адрес блока памяти
OAH - неправильная среда MS DOS
OBH - неправильный формат

0CH - неправильный код доступа
ODH - неправильные данные
OFH - дисковод указан неправильно
ЮН - попытка удалить текущий каталог
11H - задано другое устройство, что недопустимо в перемещении файлов
12H - нет больше файлов
20H - нарушение режима совместного использования файлов
21H - нарушение режима блокировки файлов
80H - файл уже используется
81H - нет больше доступных индексов файлов
82H - нет доступа для открытия файлов
83H - ошибка ввода/вывода на сетевом устройстве
84H - нет доступа для создания файлов или каталогов
85H - нет доступа для создания и удаления файлов и каталогов
86H - создаваемый файл существует, и для него установлен режим "только чтение"
87H - недопустимое использование символов шаблона в имени создаваемого файла
88H - неправильный индекс файла
89H - нет доступа для поиска
8AH - нет доступа для удаления
8BH - нет доступа для переименования
8CH - нет доступа для изменения
8DH - файлы уже используются
8EH - файлы не используются
8FH - есть файлы, для которых установлен режим "только чтение"
90H - нет файлов, для которых установлен режим "только чтение"
91H - при переименовании оказалось, что файлы с новым именем уже существуют
92H - не найдены файлы, имена которых должны быть изменены
93H - нет доступа для чтения
94H - нет доступа на запись, или для файла установлен режим "только чтение"
95H - файл отсоединен
96H - недостаточно памяти на сервере
97H - нет места на диске для файла спулинга
98H - указанный том не существует
99H - переполнение каталога
9AH - при переименовании задано новое имя для тома
9BH - неправильный индекс каталога
9CH - неправильно указан путь к каталогу
9DH - нет свободных индексов каталогов
9EH - неправильное имя файла
9FH - указанный каталог уже существует
AOH - каталог не пуст
A1H - ошибка ввода/вывода при обращении к каталогу
A2H - попытка прочитать файл с заблокированными записями
COH - нет доступа к средствам учета работы пользователей
C1H - подключение к серверу запрещено

C2H - истек предоставленный кредит
C5H - блокировка при попытке подобрать пароль
C6H - нет прав оператора консоли
D0H - ошибка при работе с очередями
D1H - нет очередей
D2H - нет сервера очередей
D3H - нет прав доступа к очереди
D4H - переполнение очереди
D5H - нет очереди заданий
D6H - нет доступа к заданию
D7H - пароль не является уникальным
D8H - слишком короткий пароль
D9H - подключение запрещено
DAH - в этот период времени подключение запрещено
DBH - с этой станции подключение запрещено
DCH - пользователь заблокирован
DEH - пароль устарел
DFH - пароль устарел
EFH - задано неправильное имя объекта
F0H - нельзя использовать символы шаблонов
F1H - неправильный код доступа
F2H - нет доступа на чтение данных об объекте
F3H - нет доступа на изменение имени объекта
F4H - нет доступа на удаление объекта
F5H - нет доступа на создание объекта
F6H - нет доступа на удаление записи
F7H - нет на создание записи, указанный диск не является локальным
F8H - объект уже подключен к серверу, нет доступа на изменение содержимого записи, объект не подключен к серверу
F9H - нет доступных каналов для подключения, нет доступа на чтение содержимого записи
FAH - нет свободных слотов для подключения сервера
FBH - неправильно заданы параметры, нет такой записи, неправильный код запроса
FCH - неизвестный файл-сервер, переполнение очереди сообщений, нет такого объекта
FDH - поле уже заблокировано
FEH - база данных объектов заблокирована, каталог заблокирован, неправильная длина имени семафора, подключение к серверу запрещено супервизором
FFH - неправильное смещение записи, ошибка в расширении имени файла, ошибка в имени файла, сбой аппаратных средств, неправильный номер дисковода, неправильное начальное значение семафора, неправильный индекс семафора, файлы не найдены, нет отклика от файл-сервера, нет такого объекта, путь не найден, сбой при работе с базой объектов, больше нет файлов, соответствующих шаблону, записи не найдены.

Глава 22. О том, какая в MS DOS имеется память и как ее использовать.

- Все говорят пет правды на земле, по правды пет и выше!

Моцарт и Сальери
ПушкинА. С.

Проблема различных видов памяти существует в MS DOS и является следствием того, что данная операционная система работает в реальном режиме, а также некоторой непродуманности ее развития. Изначально эта операционная система была задумана как однозадачная среда[61]. Именно здесь я вижу изъян, который и привел в конечном итоге к тем сложностям, которые постоянно сопровождали развитие данной операционной системы. Справедливости ради следует отметить и лепту, которую внесла в эту проблему фирма Intel с ее двухкомпонентными адресами.

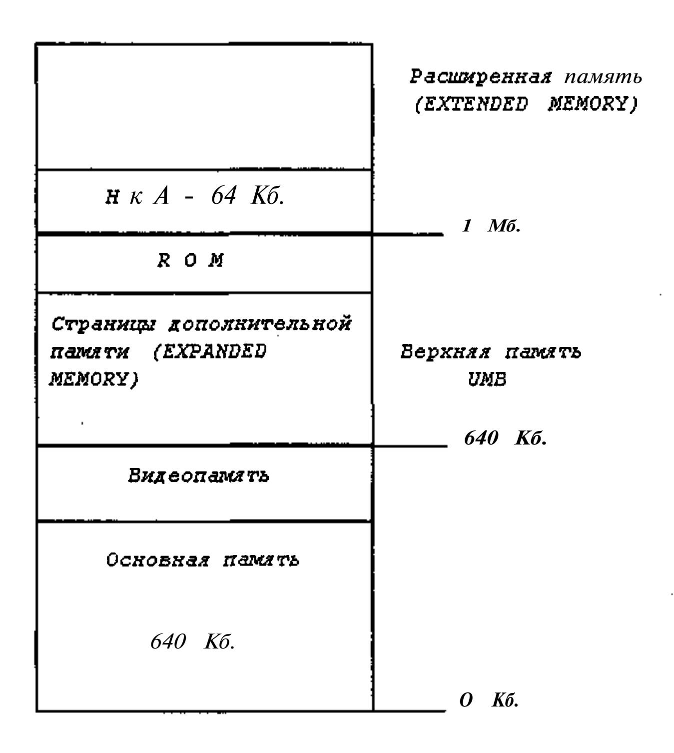


Рис. 22.1. Карта памяти персонального компьютера.

[61] Хотя в основу ОС MS DOS и были положены некоторые идеи из ОС UNIX. Это отражает некоторую недальновидность аналитиков фирмы Microsoft.

На Рис. 22.1 представлена схема различных видов памяти, которая может присутствовать у IBM-подобного компьютера в среде MS DOS. Этот рисунок следует сравнить с Рис. 2.3 (глава 2). Перечислю эти виды памяти: основная, или базовая, память, дополнительная память, верхняя память, расширенная память, HMA - High Memory Area - область старшей памяти - представляет собой фактически первые 64Кб расширенной памяти.

I. Дополнительная память.

Задолго до появления 286-х компьютеров стало ясно, что 640 Кб памяти являются серьезным препятствием для развития программного обеспечения. В начале 1985 года три фирмы: LOTUS, INTEL, MICROSOFT - выпустили спецификацию, позволяющую использовать на компьютерах XT дополнительную память. Эта спецификация получила название LIM EMS (LIM - по названию фирм, т.е. Lotus-IBM-Microsoft, EMS - EXPANDED MEMORY SPECIFICATION). В настоящее время используется спецификация версии 4.0, которая может поддерживать до 32 Мб памяти.

Установка дополнительной памяти происходила в два этапа. В свободный слот материнской платы устанавливалась плата с дополнительным ОЗУ. Для работы с этой платой должен был устанавливаться драйвер EMM. EMM делит всю ОЗУ на плате на ряд блоков по 16Кб каждый, называемых логическими страницами. Затем он находит неиспользуемый участок памяти размером 64Кб в области между 640 Кб и 1M, делит его на четыре физические страницы по 16 Кб каждая. Эта область размером 64Кб называется страничным блоком.

Чтобы получить доступ к дополнительной памяти, необходимо:

1. Запросить у EMM определенное количество логических страниц.
2. Если такое количество страниц существует, то EMM резервирует эти страницы и возвращает их дескриптор. Данный набор логических страниц в дальнейшем определяется данным дескриптором.
3. Поставить в соответствие логические и физические страницы - отобразить логические страницы на физические.
4. Поскольку адресное пространство, где помещаются физические страницы, находится в пределах 1 Мб, то доступ к дополнительной памяти осуществляется обычными командами микропроцессора.

Отображая на физические страницы те или иные логические страницы, вы можете получить доступ ко всей дополнительной памяти.

Дополнительная память необязательно должна быть установлена на плате[62]. На компьютерах, начиная с 80286-го, появилась расширенная память (см. ниже). Специальные драйверы могут конвертировать расширенную память в дополнительную память. На AT386 и выше это такие программы, как EMM386.EXE и пакет QEMM, которые, предоставляя сеанс MS DOS, также эмулируют рассматриваемые виды памяти.

---

[62] Автор, к сожалению, не встречал компьютеры с дополнительной памятью на плате.

П. Интерфейс с драйвером EMM. (EXPANDED MEMORY MANAGER)

Связь с драйвером осуществляется через прерывание 67H. В регистр АН следует поместить номер функции. После выполнения функции в АН находится результат выполнения. Если возвращается 0, то функция выполнена успешно. Здесь перечислены не все функции драйвера, но их более чем достаточно для программного использования.

1. Состояние драйвера EMM.
   Вход:
   AH - 40H
   Выход:
   AH - OOH память готова к работе,
   80H внутренняя ошибка,
   81H аппаратная ошибка.
2. Чтение адреса страничного окна.
   Вход:
   AH-41H
   Выход:
   AH - OOH нормальное завершение,
   BX - сегментный адрес страничного окна,
   AH - 80H внутренняя ошибка,
   AH - 81H аппаратная ошибка.
3. Чтение числа страниц EMS памяти.
   Вход:
   AH - 42H
   Выход:
   AH - OOH нормальное завершение,
   BX - количество свободных страниц,
   DX - общее количество страниц EMS памяти,
   AH - 80H внутренняя ошибка,
   AH - 81H аппаратная ошибка.
4. Выделение страниц EMS памяти.
   Вход:
   AH - 43H
   BX - число требуемых логических страниц.
   Выход:
   AH - OOH нормальное завершение,
   DX - код описателя для доступа к выделенным страницам,
   AH - 80H внутренняя ошибка,
   AH - 81H аппаратная ошибка,
   AH - 85H свободных описателей нет,
   AH - 87H недостаточно свободных страниц,
   AH - 88H запрос на выделение страниц отсутствует.

5. Задание отображения.
Вход:
AH - 44H
AL - номер физической страницы (0-3),
BX - номер логической страницы (нумерация также с 0),
DX - описатель.
Выход:
AH - OOH нормальное завершение,
AH - 80H внутренняя ошибка,
AH - 81H аппаратная ошибка,
AH - 83H неверный описатель,
AH - 8AH неверная логическая страница,
AH - 8BH неверная физическая страница.

6. Освобождение страниц EMS.
Вход:
AH - 45H
DX - описатель.
Выход:
AH - OOH нормальное завершение,
AH - 80H внутренняя ошибка,
AH - 81H аппаратная ошибка,
AH - 83H неверный описатель,
AH - 85H ошибка при сохранении и восстановлении.

7. Определение версии драйвера EMM.
Вход:
AH - 46H
Выход:
AH - OOH нормальное завершение,
AL - номер версии драйвера EMM,
AH - 80H внутренняя ошибка,
AH - 81H аппаратная ошибка.

8. Запоминание карты отображения.
Вход:
AH - 47H
DX - описатель.
Выход:
AH - OOH нормальное завершение,
AH - 80H внутренняя ошибка,
AH - 81H аппаратная ошибка,
AH - 83H неверный описатель,
AH - 8EH память карты отображения не содержит записи для данного описателя.

9. Восстановление карты отображения.
Вход:
AH - 48H
DX - описатель.

Выход:
AH - OOH нормальное завершение,
AH - 80H внутренняя ошибка,
AH - 81H аппаратная ошибка,
AH - 83H неверный описатель,
AH - 8EH память карты отображения не содержит записи для данного описателя.

10. Чтение числа описателей.

Вход:
AH - 4BH.
Выход:
AH - OOH нормальное завершение,
BX - число выделенных описателей,
AH - 80H внутренняя ошибка,
AH - 81H аппаратная ошибка.

11. Чтение числа выделенных страниц.

Вход:
AH - 4CH
DX - описатель.
Выход:
AH - нормальное завершение,
BX - число выделенных страниц,
AH - 80H внутренняя ошибка,
AH - 81H аппаратная ошибка.
AH - 83H неверный описатель.

12. Чтение информации обо всех описателях.

Вход:
AH - 4DH
ES - адрес сегмента массива,
DI - смещение массива.
Выход:
AH - OOH нормальное завершение,
BX - число выделенных логических страниц,
AH - 80H внутренняя ошибка,
AH - 81H аппаратная ошибка.

В случае нормального завершения область указанной памяти будет заполнена информацией. Информация будет состоять из записей. Каждая запись - это два слова, первое слово - описатель, второе - количество выделенных страниц.

Ниже представлена программа, демонстрирующая работу с дополнительной памятью.

```
DATA   SEGMENT
TEXT1  DB 'Ошибка при работе с драйвером EMM!',13,10,'$'
TEXT2  DB 'Недостаточное количество страниц.',13,10,'$'
```

```
STRING DB  'Эта строка для демонстрации работы со'
      DB 'страницами памяти EMS.',13,10,'$'
DATA ENDS
_ST SEGMENT STACK
      DB 100 DUP(?)
_ST ENDS
CODE SEGMENT
      ASSUME CS:CODE, DS:DATA, SS:_ST
BEG:
;начальная установка регистров
      MOV AX,DATA
      MOV DS,AX
;установить флаг обработки строк
      CLD
;проверка наличия драйвера EMM
      MOV AH,40H
      INT 67H
      OR  AH,AH
      JZ  OK1
      CALL ERR
      JMP  DOS
OK1:
/чтение адреса сегмента страничного окна
      MOV  AH,41H
      INT  67H
      OR   AH,AH
      JZ   OK2
      CALL ERR
      JMP  DOS
OK2:
      MOV  ES,BX ;сегментный адрес EMS памяти
/чтение числа страниц EMS памяти
      MOV  AH,42H
      INT  67H
      OR   AH,AH
      JZ   OK3
      CALL ERR
      JMP  DOS
OK3:
      CMP  BX,2
      JNB  OK4
      LEA  DX,TEXT2
      CALL PRINT
      JMP  DOS
```

```
OK4:
;выделяем одну страницу
      MOV   AH,43H
      MOV   BX,1
      INT   67H
      OR    AH,AH
      JZ    OK5
      CALL  ERR
      JMP   DOS
OK5:
      MOV   BP,DX ;сохраним описатель
;отображаем страницу на физическую область
      MOV   AH,44H
      MOV   AL, O
      MOV   BX,0
      INT   67H
      OR    AH,AH
      JZ    0K6
      CALL  ERR
      JMP   DOS
OK6:
;копируем строку STRING на страницу O
      LEA   SI,STRING
      XOR   DI,DI
LOO:
      MOV   AL,DS:[SI]
      MOV   ES:[DI],AL
      INC   SI
      INC   DI
     CMPAL,'$'
      JNZ   LOO
;выделяем еще одну страницу
      MOV AH,43H
      MOV BX,1
      INT 67H
      OR  AH,AH
      JZ  OK7
      CALL  ERR
      JMP DOS
OK7:
;отображаем ее
      MOV AH,44H
      MOV AL,1
      MOV BX,0
```

```
        INT  67H
        JZ   OK8
        CALL ERR
        JMP  DOS
OK8:
;копируем первую страницу на вторую
;ES-на страницу 1, DS-на страницу 0
        MOV AX,ES
        MOV DS,AX
        ADD AX,1024  ;начало страницы 1
        MOV ES,AX
        XOR SI,SI
        XOR DI,DI
        MOV CX,16*1024
L001:
        MOV AL,DS:[SI]
        MOV ES:[DI],AL
        INC SI
        INC DI
        LOOP  L001
;освобождаем первую страницу  (первый описатель)
        XCHG DX,BP
        MOV   AH,45H
        INT   67H
        OR    AH,AH
        JZ    OK9
        CALL  ERR
        JMP   DOS
OK9:
        MOV   DX,BP
;отображаем  страницу на вторую физическую страницу
        MOV   AH,44H
        MOV   AL,2
        MOV   BX,0
        INT   67H
        JZ    OK10
        CALL  ERR
        JMP   DOS
OK10:
;проверяем  содержимое
        PUSH DX
        MOV   DX,32*1024
        CALL  PRINT
        POP   DX
```

```
;закрываем страницы
CLOSE:
        MOV  AH,45H
        INT  67H
;выход в DOS
DOS:
        MOV  AH,4CH
        INT  21H
;область процедур
;процедура вывода сообщения об ошибке
ERR     PROC
        PUSH DS
        PUSH AX
        LEA  DX,TEXT1
        MOV  AX,DATA
        MOV  DS,AX
        CALL PRINT
        POP  AX
        POP  DS
        RET
ERR     ENDP
;процедура вывода строки на экран
PRINT  PROC
        MOV AH,9
        INT 21H
        RET
PRINT  ENDP
CODE  ENDS
        END BEG
```

РМС. 22.2. *Пример использования дополнительной памяти.*

В программе на Рис. 22.2 есть один недостаток: она не проверяет, установлен ли вообще драйвер EMM. Рассмотрим теперь, как это делается. Алгоритм обнаружения драйвера EMM основан на том, что если драйвер установлен, то в системе появляется устройство с именем "EMMXXXX0". Поэтому можно попытаться открыть файл с таким именем при помощи функции 3DH. Если открытие произошло, то теперь нужно проверить, нет ли на диске такого файла. Для этого можно воспользоваться функцией DOS 44H - ее подфункцией 0. Кроме того, необходимо проверить, доступно ли устройство для ввода. Это можно сделать при помощи подфункции 7 той же функции 44H. Ниже (Рис. 22.3) представлена простая программа, правильно определяющая присутствие драйвера EMM.

```
CODE   SEGMENT
        ASSUME CS:CODE,DS:CODE
        ORG 100H
```

```
BEGIN:
;открыть устройство EMMXXXX0
      MOV AH,3DH
      MOV AL,0
      LEA DX,STR_EMM
      INT 21H
      JC  NO_EMM1
;проверить, нет ли файла с таким именем
      MOV BX,AX
      MOV AX,4400H
      INT 21H
      JC  NO_EMM
;присутствие 7-го бита означает, что это не файл
      TEST DX,80H
      JZ  NO_EMM
;проверить статус устройства
      MOV AX,4407H
      INT 21H
      JC  NO_EMM
      CMP AL,0
      JZ  NO_EMM
      LEA DX,STR2
      JMP SHORT KONEC
NO_EMM:
      LEA DX,STR1
KONEC:
      MOV AH,9
      INT 21H
;закрыть устройство
      CALL CLOSE
      RET
NO_EMM1:
      LEA DX,STR1
      MOV AH,9
      INT 21H
      RET
;блок сообщений
STR_EMM  DB "EMMXXXX0",0
STR1     DB "Драйвер EMM отсутствует", 13,10,'$'
STR2     DB "Драйвер EMM присутствует", 13,10, '$'
;процедура закрытие устройства или файла
CLOSE PROC
      MOV AH,3EH
      INT 21H
      RET
```

```
CLOSE  ENDP
CODE       ENDS
       END  BEGIN
```

Рис. 22.3 Программа, определяющая присутствие драйвера EMM.

Другой способ проверки присутствия драйвера EMM в системе основан на том, что вектор 67H должен указывать в тело драйвера. Более того, на расстоянии 10 байт в сторону старших адресов от точки, куда указывает вектор, располагается строка 'EMMXXXXO'. Проверка наличия такой строки и будет являться проверкой наличия в системе драйвера. Однако этот способ не слишком надежен, так как ничто не мешает какой-нибудь программе перенаправить этот вектор на свою подпрограмму.

III. Расширенная память.

Расширенная память - это память с адресами свыше одного мегабайта. Мы сталкивались с ней в главах 5 и 20 (см. Рис. 5.2 и 20.2). В главе 5 был показан доступ к этой памяти посредством прерывания 15H, а в главе 20 — непосредственным переходом в защищенный режим. Драйвер HIMEM.SYS также предоставляет возможности работы с расширенной памятью. Это интерфейс более высокого уровня, потому что позволяет использовать расширенную память совместно нескольким программам. Доступ к расширенной памяти посредством прерывания 15H считается в настоящее время устаревшим. Драйвер HIMEM.SYS блокирует этот доступ. Ключ INT15=N, где N - объем расширенной памяти, доступ к которой можно получить посредством прерывания 15H, позволяет разблокировать его для Ваших программ.

IV. Область HMA.

Область HMA (HIGH MEMORYAREA) расположена сразу за системным ROM BIOS (выше 1 Мб) и имеет размер приблизительно 64 Кб. Своему существованию она целиком обязана эмуляции процессором 80286 процессора 8088. Максимальный полный адрес, по которому может обратиться процессор 8088, составляет FFFF:000F, что соответствует 20 адресным линиям. Если увеличить это значение на 1, то произойдет циклический перенос, и значение адреса станет 0000:0000. Таким образом, для процессора 8088 память с адресами FFFF:0010 - FFFF:FFFF (это 64Кб без 16 байт) становится невидимой. Для процессора 80286 эта проблема может быть решена путем использования адресной линии А20. Для совместимости, однако, процессор 80286 (а также 80386 и т.д.) в обычном состоянии (отключена адресная линия А20) ведет себя так же, как и 8088.

HMA фактически является началом расширенной памяти. Доступ к HMA корректно можно осуществить, используя интерфейс драйвера HIMEM.SYS (см. ниже). Начиная с версии 5.0, MS DOS умеет забрасывать в HMA часть своих файлов (DOS=HIGH).

V. Верхняя память (UMB).

UMB - сокращение английского названия UPPER MEMORY BLOKS, т.е. блоки верхней памяти. Как и откуда она может появиться? Область памяти между 640Кб и 1 Мб зарезервирована для видеобуферов и ПЗУ. Однако многие блоки в этой области

оказываются неиспользуемыми. В частности, именно сюда помещается дополнительная память.

Такие драйверы, как EMM386.EXE, QEMM и др., переводят компьютер в виртуальный режим. В этом режиме (он существует, начиная с 386-го процессора) посредством страничной организации можно перенаправить операции чтения или записи из одной области пространства в другую. Таким образом, появляется возможность загружать резидентные программы и драйверы в UMB, тем самым освобождая основную память.

VI. Интерфейс с драйвером HIMEM.SYS.

Драйвер HIMEM.SYS предоставляет возможность работать с тремя видами памяти: расширенной, HMA и UMB. Для того чтобы использовать возможности драйвера, программа должна:

1. Определить его наличие в памяти. Это осуществляется посредством прерывания 2FH:

```
MOV AX,4300H
INT  2FH
CMP AL,80H  ;если в AL 80H, то драйвер установлен
JNE  NO_HIM
```

2. Определить 32-разрядный адрес точки входа драйвера:

```
.
ADR_HIM_OFF DW ?
ADR_HIM_SEG DW ?
.
MOV AX,4310H
INT  2FH
MOV  WORD  PTR  CS:ADR_HIM_OFF,BX
MOV  WORD  PTR  CS:ADR_HIM_SEG,ES
.
```

Вызов функции драйвера осуществляется командами:

```
MOV AH,8 ;номер функции
CALL  DWORD  PTR  CS:ADR_HIM_OFF ;длинный вызов
```

Может возникнуть вопрос, почему вызов функции не осуществить посредством того же прерывания 2FH. Но дело в том, что данное прерывание может использоваться многими программами, и, следовательно, выполнение его может сильно замедлиться.

3. Драйвер предоставляет возможность работать с 18 функциями, из которых 8 предназначены для работы с расширенной памятью, по 2 - для работы с HMA и UMB, а также 6 дополнительных функций. Вот эти функции.

Номер версии.
Вход: AH - 0H
Выход: AX - номер версии,
BX - внутренний номер,
DX - флаг, если есть HMA.

Выделение HMA.
Вход: AH - 1H,
DX - размер в байтах
Выход: AX - 1, если успешно

Освобождение HMA.
Вход: AH - 2H
Выход: AX - 1, если успешно

Глобальное разрешение A20
Вход: AH - 3H
Выход: AX - 1, если успешно

Глобальное запрещение A20
Вход: AH - 4H
Выход:

Локальное разрешение A20.
Вход: AH - 5H
Выход: AX - 1, если успешно

Локальное запрещение A20.
Вход: AH - 6H
Выход:

Запрос состояния A20.
Вход: AH - 7H
Выход: AX - флаг,
BL - код ошибки

Запрос свободных блоков расширенной памяти.
Вход: AH - 8H
Выход: AX - наибольший свободный блок
DX - общий размер свободной памяти.

Выделение свободных блоков расширенной памяти.
Вход: AH - 9H
DX - размер памяти в Кб.
Выход: DX - описатель

Освобождение расширенной памяти.
Вход: AH - 0AH
DX - описатель
Выход:

Копирование в расширенную память.
Вход: AH - 0BH
DX - описатель
Выход: DS:SI - указатель на структуру

Смещение	Тип	Содержание
0	DWORD	Размер
4	WORD	Дескриптор источника
6	DWORD	Адрес источника
10	WORD	Дескриптор источника
12	DWORD	Адрес источника

Блокирование расширенной памяти.
Вход: AH - 0CH
DX - описатель
Выход: DX:BX - 32-разрядный адрес блока

Разблокирование расширенной памяти.
Вход: AH - 0DH
DX - описатель
Выход:

Информация о расширенной памяти.
Вход: AH - 0EH
DX - описатель
Выход: DX - размер блока в Кб,
BH - число запретов на блок,
BL - число еще свободных описателей

Изменение размера расширенной памяти.
Вход: AH - 0FH
DX - описатель
Выход: BX - новый размер в Кб,
DX - описатель

Выделение UMB.
Вход: AH - 10H
DX - необходимый размер в параграфах.
Выход: BX - сегмент,
DX - действительный размер в параграфах.

Освобождение UMB.
Вход: AH - 11H
BX - сегмент.
Выход:

Большинство указанных функций возвращает в регистр AX код результата. Если функция выполнена успешно, то в регистр AX помещается 1, если нет, то 0. Данные функции эмулируются драйвером QEMM с полностью идентичным доступом.

Ниже приведен пример использования HMA. Если память свободна (там может находиться часть MS DOS), то Вы можете использовать ее в своих программах, например, для хранения данных.

```
DATA SEGMENT
TEXT1 DB 'Драйвер не установлен.',13,10, '$'
TEXT2 DB 'Нехватка HMA',13,10,'$'
TEXT3 DB 'Проверка!',13,10,'$'
DATA ENDS
STT SEGMENT STACK
      DB 100 DUP(?)
STT ENDS
CODE SEGMENT
      ASSUME CS:CODE, DS:DATA, SS:STT
BEGIN:
;--сегмент данных
      MOV AX,DATA
      MOV DS,AX
;--определяем наличие драйвера
      MOV AX,4300H
      INT 2FH
      CMP AL,80H
      JNE NO_HIM
      JMP SHORT YES
NO_HIM:
      LEA DX,TEXT1
      MOV AH,9
      INT 21H
      JMP KON
YES:
;--определяем точку входа
      MOV AX,4310H
      INT 2FH
      MOV WORD PTR CS:ADR_HIM_OFF,BX
      MOV WORD PTR CS:ADR_HIM_SEG,ES
;--запрашиваем HMA
      MOV  AH,1
      MOV  DX,20   ;просим 20 байт
      CALL DWORD PTR CS:ADR_HIM_OFF
      CMP  AX, 1
      JNZ   NO_OK
```

```
;-- разрешаем A20
       MOV  AH, 3
       CALL DWORD PTR CS:ADR_HIM_OFF
       CMP  AX, 1
       JNZ  NO_OK
       JMP  SHORT OK
NO_OK:
;--память выделить не удалось
       LEA DX,TEXT2
       MOV AH,9
       INT 21H
;освобождаем HMA .
       MOV AH,2H
       CALL DWORD PTR CS:ADR_HIM_OFF
;запрещаем A20
       MOV  AH, 4
       CALL DWORD PTR CS:ADR_HIM_OFF
       JMP  KON
OK:
;--копируем строку из DATA в память сразу за 1MB
       MOV  AX,0FFFFH
       MOV  ES,AX
       LEA  SI,TEXT3
       MOV  DI,010H
LOO:
       MOV  AL, [SI]
       MOV  ES:[DI],AL
       CMP  AL,'$'
       JZ   ALL
       INC  DI
       INC  SI
       JMP  SHORT LOO
ALL:
;--теперь печатаем
       MOV DX, 010H
       MOV AX,ES
       MOV DS,AX
;--DS:DX указывает на память сразу за 1MB
       MOV AH,9
       INT 21H
;--освобождаем HMA
       MOV BX,ES
       MOV AH,2H
       CALL DWORD PTR CS:ADR_HIM_OFF
```

```
;--запрещаем A20
       MOV  AH,4
       CALL DWORD PTR CS:ADR_HIM_OFF
KON:
       MOV AH,4CH
       INT 21H
;----------
ADR_HIM_OFF DW ?
ADR_HIM_SEG DW ?
CODE ENDS
       END BEGIN
```

***Рис. 22.4** Простой пример использования HMA.*

Программа на Рис. 22.4 копирует строку из сегмента данных в область HMA, а затем печатает строку уже из этой области.

В заключение обращаю внимание читателей на тот факт, что Windows 95 (и 98) эмулирует те виды памяти, которые мы сейчас разобрали для запускаемых из нее программ, предназначенных для запуска в среде MS DOS.

Глава 23. Тестирование оборудования.

Доверяй, но проверяй.

Русская пословица.

К проблеме тестирования оборудования я неоднократно обращался в предыдущих главах. В данной главе будут рассмотрены некоторые вопросы, не нашедшие своего раскрытия ранее. Надо сказать, что тестирование оборудования является значительной проблемой программирования вообще. Особенно это актуально для такой операционной системы, как MS DOS. Практически не занимаясь управлением внешних устройств, она оставляет это на совести (правильнее сказать на компетентности) программистов. Особенно большие проблемы возникают с графическими устройствами. Я думаю, что многие читатели сталкивались с проблемой, когда игровые программы по непонятным причинам отказывались работать с некоторыми графическими системами. Иная ситуация с операционной системой Windows. Она берет на себя (в отличие от MS DOS) практически все взаимодействие с внешними устройствами, и, для того чтобы знать о наличии или отсутствии того или иного устройства, необходимо лишь осуществить тот или иной системный вызов.

Глава не претендует на полноту изложения, скорее, это несколько примеров того, как пользовательская программа может взаимодействовать с компьютерным оборудованием.

I. Определение типа микропроцессора.

В главе 4 были приведены некоторые критерии, по которым можно различать микропроцессоры INTEL разных поколений. Здесь приводится программа, позволяющая однозначно определить вид микропроцессора. При написании программы автором были заимствованы отдельные идеи из [15].

```
;основной модуль программы TEST
.8086
EXTRN TO_386:FAR .
PUBLIC TEXT
PUBLIC TX_80386
PUBLIC TX_80486
PUBLIC TX_80586
PUBLIC VIRT
DATA SEGMENT PARA
;блок сообщений
TX_8088  DB 'Это процессор 8088',13,10,'$'
TX_8086 DB 'Это процессор 8086',13,10, ' $ '
TX_NEC20 DB 'Это процессор NEC20',13,10,'$'
TX_NEC30 DB 'Это процессор NEC30',13,10, ' $ '
TX_80188 DB 'Это процессор 80188',13,10, ' $ '
```

```
TX_80186 DB 'Это процессор 80186',13,10,'$'
TX_80286 DB 'Это процессор 80286',13,10,'$'
TX_80386 DB 'Это процессор 80386',13,10,'$'
TX_80486 DB 'Это процессор 80486',13,10,'$'
TX_80586 DB 'Это процессор Pentium',13,10,'$'
VIRT DB 'Виртуальный режим.',13,10, '$'
DATA ENDS
ST1 SEGMENT PARA STACK 'STACK'
      DW 50 DUP(?)
ST1 ENDS
CODE SEGMENT PARA PUBLIC
      ASSUME CS:CODE, DS:DATA, SS:ST1
BEGIN:
      MOV AX,DATA
      MOV DS,AX
      PUSH SP
      POP AX
      CMP SP,AX
      JNZ BELOW_286
;здесь 286 и выше
      MOV  AX,7000H
      PUSH AX
      POPF
      PUSHF
      POP AX
      TEST AX,7000H
      JNZ  NO_286
;286-й
      LEA DX,TX_80286
      CALL FAR PTR TEXT
      JMP _END
N0_286:
      JMP TO_386
;здесь 8086,8088,80186,80188,NEC20,NEC30
BELOW_286:
      CALL TBUFFER
      MOV  BP,CX
      MOV  CL,33
      MOV  AX,0FFFFH
      SHL  AX,CL
      JZ   NO_186
      CMP  BP,0
      JNZ  _188
      LEA DX,TX_80186
```

```
        CALL  FAR  PTR  TEXT
        JMP  SHORT  _END
_188:
        LEA  DX,TX_80188
        CALL  FAR  PTR  TEXT
        JMP  SHORT  _END
NO_186:
        MOV  CX,0FFFFH
        JMP  SHORT  $+2
        DB  0F3H,026H,0ACH   ;REP LODSB E:
        JCXZ  NEC
;8088/8086
        CMP  BP,0
        JNZ   _88
        LEA  DX,TX_8086
        CALL  FAR  PTR  TEXT
        JMP  SHORT  _END
_88:
        LEA  DX,TX_8088
        CALL  FAR  PTR  TEXT
        JMP  SHORT  _END
NEC:
        CMP  BP,0
        JNZ  NEC20
        LEA  DX,TX_NEC30
        CALL  FAR  PTR  TEXT
        JMP  SHORT  _END
NEC20:
        LEA  DX,TX_NEC20
        CALL  FAR  PTR  TEXT
_END:
        MOV  AH,4CH
        INT  21H
;область процедур
;процедура для тестирования длины буфера
;в CX - 0, если буфер составляет 6 байт
TBUFFER PROC NEAR
        PUSH ES
        PUSH DI
        STD
        PUSH CS
        POP  ES
        LEA  DI,CS:MET2
        MOV  AL,BYTE  PTR  CS:MET1
```

```
        MOV CX,3
        CLI
        REP  STOSB
        CLD
        NOP
        NOP
        NOP
        INC  CX
MET1:   STI
MET2:   STI
        POP  DI
        POP  ES
        RETN
TBUFFER  ENDP
TEXT  PROC  FAR
        MOV  AH,9
        INT  21H
        RETF
TEXT  ENDP
CODE  ENDS
        END  BEGIN

;второй модуль программы TEST1
.386P
PUBLIC  TO_386
EXTRN  TEXT:FAR
EXTRN  VIRT:BYTE
EXTRN  TX_80386:BYTE
EXTRN  TX_80486:BYTE
EXTRN  TX_80586:BYTE
CODE1  SEGMENT  PARA  USE16
        ASSUME  CS:CODE1
TO_386:
        MOV  EAX,CR0
        TEST  AL,1
        JZ  NO_VIRT
        LEA  DX,VIRT
        CALL    FAR  PTR  TEXT
NO_VIRT:
        AND AX,1111110001111111B
        PUSH  AX
        POPF
        CALL  M486_386
        CMP  AL,0
```

```
        JNZ    _486
        LEA  DX,TX_80386
        CALL  FAR  PTR  TEXT
        JMP _END
_486:
;здесь 486 или выше
        CALL  PENT_486
        CMP AL,0
        JNZ _PENT
        LEA  DX,TX_80486
        CALL  FAR  PTR  TEXT
        JMP   _END
_PENT:
;это Пентиум
        LEA  DX,TX_80586
        CALL  FAR  PTR  TEXT
_END:
        MOV AH,4CH
        INT 21H
;процедура различает 386-й от 486-го
;проверяется 18-й бит в регистре флагов
M486_386 PROC
        CLI
        PUSHFD
        POP   EAX
        AND    EAX,11111111111110111111111111111111B
        PUSH EAX
        POPFD
        PUSHFD
        POP EAX
        TEST  EAX,00000000000001000000000000000000B
        JNZ   NO_486
        OR     EAX,00000000000001000000000000000000B
        PUSH EAX
        POPFD
        PUSHFD
        POP EAX
        TEST  EAX,00000000000001000000000000000000B
        JZ   N0_486
        MOV AL,1
        STI
        RETN
NO_486:
        MOV AL,0
```

```
        STI
        RETN
M486_386 ENDP
;процедура, распознающая Pentium, основанная на отсутствии
;у него буфера команд
PENT_486 PROC
        MOV AL,1
        JMP $+2     ;сбросить очередь команд
        MOV byte ptr $+6,0c3h  ;код C3h <-> RET
        NOP
NO_PENT:
        MOV AL,0                     ;здесь процессор 80486
        RETN
PENT_486 ENDP
CODE1  ENDS
        END TO_386
```

Рис. 23.1. Программа, определяющая тип микропроцессора.

Данная программа состоит из двух модулей. Это не случайно. Дело в том, что часть команд должна выполняться в режиме 386-го процессора. Они вынесены в модуль TEST1. Программа определяет весь спектр существующих микропроцессоров вплоть до Pentium'a. Отличие Pentium'a от других микропроцессоров семейства Intel заключается в отсутствии у него буфера команд (см. главу 4). Обратите Ваше внимание на то, как стыкуются друг с другом модули. Здесь Вам понадобится информация из главы 13.

Далее приводится словесный алгоритм определения типа микропроцессора.

1. Начало: разбиваем все семейство микропроцессоров на то, что ниже 286-го (нижнее семейство), и остальные. Дело в том, что, начиная с 286-го микропроцессора, стек начал работать несколько по-иному. Вопрос заключается в том, когда меняется содержимое SP – до команды PUSH или после. Индикатором является команда PUSH SP. У микропроцессоров 286-х и выше содержимое SP меняется после команды PUSH.
2. Рассмотрим теперь «нижнее семейство». Вначале определяется индикатор длины буфера команд: процедура TBUFFER возвращает в CL 0, если длина буфера команд составляет 6 байт.[63] Далее используется особенность работы сдвиговых операций для микропроцессоров 80186/80188. Выделив это подсемейство, мы легко отделяем 186-й от 188-го, т.к. длина буфера 186-го составляет 6 байт.
3. Следующий этап - семейство NEC.[64] Оно определяется по особенности работы команды LODSB. Далее NEC30 отличается от NEC20 опять тем, что длина буфера у NEC30 составляет 6 байт.

---

[63] Процедура взята из [15].

[64] В отличие от подсемейства 80186/80188 автору приходилось довольно часто сталкиваться с компьютерами на базе микропроцессоров семейства NEC.

4. Теперь у нас в «нижнем семействе» осталось всего два микропроцессора: 8086 и 8088. И здесь идем по пути сравнения длин буферов команд: у микропроцессора 8086 длина составляет 6 байт.
5. Вернемся теперь к остальным микропроцессорам. Процессор 80286 легко определяем по причине того, что, хотя у него в регистре флагов и появились, биты 12, 13, 14(см. главу 20), в реальном режиме они не устанавливаются.
6. Наличие виртуального режима у микропроцессоров 386-х и выше определяем по наличию нулевого бита в регистра CRO.
7. 486-й процессор отличается от 386-го по наличию 18-гобита в регистре флагов.
8. Наконец Pentium отличается от всех остальных отсутствием у него буфера команд.

II. Определение типа видеосистемы.

В главе 7 (Рис. 7.7) была приведена процедура определения типа видеоадаптера. И здесь добавить нечего.

III. Определить присутствие сопроцессора.

Программа определения присутствия сопроцессора основана на том факте, что команды сопроцессора при отсутствии такового просто игнорируются. Следовательно, достаточно выполнить несколько команд сопроцессора и проверить получившийся результат. Поскольку, начиная с 486-го микропроцессора, сопроцессор стал его неотъемлемой частью, приведенная ниже программа носит в значительной степени познавательный характер.

```
CODE   SEGMENT
       ASSUME CS:CODE, DS:CODE
       ORG  100H
BEGIN:
;готовим данные
       MOV TEST1,7890H
       MOV TEST2,0
;выполнить команды сопроцессора
;поместить в вершину стека
       FILD TEST1
;взять с вершины стека
       FIST TEST2
;сравниваем
       MOV  AX,TEST1
       CMP  AX,TEST2
       JZ   YES
;сопроцессор отсутствует
       LEA  DX,MES2
       JMP  SHORT _END
YES:
```

```
;сопроцессор присутствует
      LEA  DX,MES1
_END:
      MOV AH, 9
      INT  21H
      RET
;данные
TEST1  DW  ?
TEST2  DW  ?
MES1  DB  'Сопроцессор присутствует.', 13,10,'$'
MES2  DB  'Сопроцессор не обнаружен.', 13,10,'$'
CODE  ENDS
      END  BEGIN
```

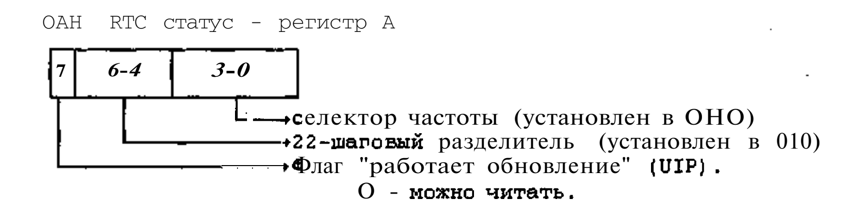
Рис. 23.2. Программа определения присутствия сопроцессора.

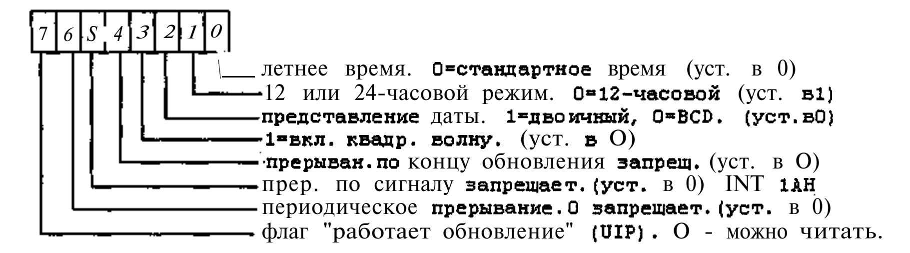
IV. CMOS-память.

Компьютеры AT имеют питаемые от батарейки часы реального времени (RTC - Real-Time Clock) и CMOS-память (64 байта). Эта память содержит разнообразную информацию о данном компьютере. Доступ к CMOS-памяти осуществляется через 70H, 71H порты. Вначале в порт 70H помещается адрес ячейки памяти, а затем через порт 71H осуществляется чтение или запись. Ниже дана расшифровка содержимого ячеек, взятых мною из "Электронного справочника программиста".

Адрес	Описание
0	Текущая секунда часов
1	Сигнальная секунда
2	Текущая минута
3	Сигнальная минута
4	Текущий час
5	Сигнальный час
6	Текущий день недели (1 - воскресенье)
7	Текущий день месяца
8	Текущий месяц
9	Текущий год (две цифры)

Все порции RTC хранятся в формате BCD как две десятичные цифры; например, 31 хранится как 31H.





ОСН RTC статус - регистр С. Биты статуса прерывания можно только читать.

Биты:

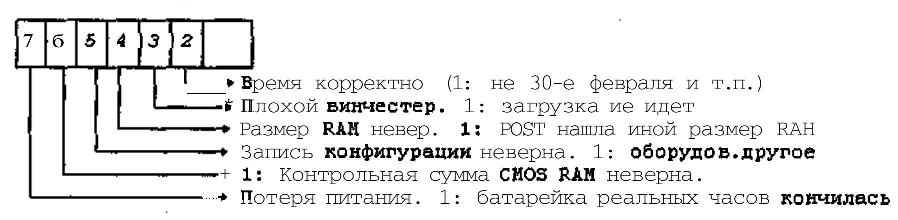
0-3 - резерв,

4,5,6 - сигнализируют об окончании прерывания (одного из трех видов, см. Приложение 9),

7 - указывает на наличие прерывания (1).

Обработчик прерывания должен прочесть этот регистр, иначе следующее прерывание не будет выработано.

0DH RTC статус - регистр D. Бит 7=1: CMOS-RAM получает питание
=0: батарейка села.



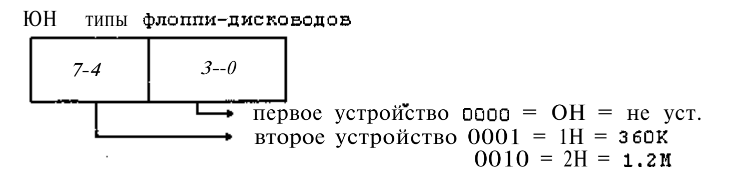
0FH байт статуса закрытия

Этот байт считывается после сброса CPU, чтобы определить, вызван ли сброс с целью выйти из защищенного режима работы процессора 80286.

0 = мягкий сброс (CTRL-ALT-DEL) (или неожиданный). Обойти POST

1 = закрытие после определения размера памяти

2 = закрытие после выполнения теста памяти
3 = закрытие после ошибки памяти (сбой четности 1 или 2)
4 = закрытие по запросу начального загрузчика
5 - закрытие по FAR JMP (рестарт контроллера прер. и JMP 0:[0467H])
6, 7, 8 = закрытие после прохода теста защищенного режима
9 = закрытие после пересылки блока. См. INT 15H подф. 87H
OAH = закрытие по FAR JMP (немедленный JMP по адресу в 0:[0467H])

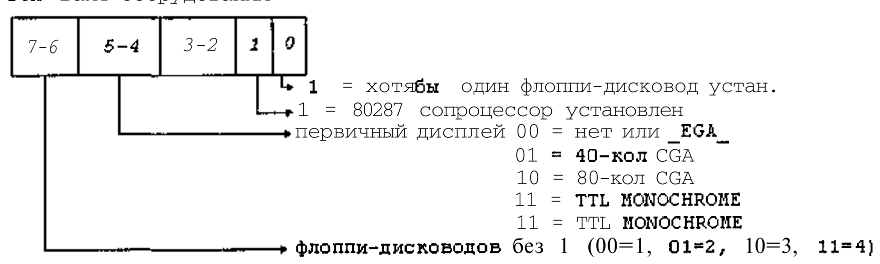


11H резерв

```
12H  тип винчестера (для устройств C: и D:, если от 1 до 14)
┌────────┬────────┐
│  7-4   │  3-0   │
└────────┴────────┘
     │         └──►первый винчестер (устр. C:)    0000 =отсут.
     └───────────────► второй винчестер (устр. D:)  иначе =ID типа (ниже)
1111    =исп.адр.19H/1AH
```

См. Типы винчестеров AT BIOS об устройствах, поддерживаемых BIOS.

13H резерв



15H основная память (младш) ----> 0100H=256K, 0200H=512K, 0280H=640K
16H основная память (старш)
17H расширенная память за 1M (младш)——>(в K-байтах. 0-3C00H)
18H расширенная память (старш)———> См. INT 15H подф. 88H
19H диск 0 (устр. C:) тип винчестера, если (CMOS ADDR 12H & OFH) = OFH
1AH диск 1 (устр. D:) тип винчестера, если (CMOS ADDR 12H & FOH) = FOH
1BH-2DH резерв
2EH контр. сумма по адресам CMOS 10H..2DH (старший байт)
2FH (младший байт)
3OH расширенная память за 1M (младш) ----> (в K-байтах. 0-3COOH)

31H расширенная память (старш)———> См. INT 15H подф. 88H

32H столетие в коде BCD (например, 19H)

33H смешанная информация. Бит7=IBM 128K необязательная плата памяти

Бит 6=используется утилитой "SETUP" 34H-3FH резерв.

Ниже мы представляем программу проверки контрольной суммы CMOS. В старых справочниках говорилось, что контрольная сумма берется с ячеек 10H-20H. В новых компьютерах берутся ячейки 10H-2DH. В старых компьютерах ячейки 21H-2DH не использовались и имели нулевое значение. По этой причине программа на Рис. 23.3 должна давать правильный результат и на старых компьютерах AT.

```
CODE  SEGMENT
      ASSUME CS:CODE, DS:CODE
      ORG  100H
BEGIN:
;поместить контрольную сумму в регистр DX
      MOV AL,2EH
      OUT 70H,AL
      JMP $+2
      IN  AL,71H
      MOV DH,AL
      MOV AL,2FH
      OUT 70H,AL
      JMP $+2
      IN  AL,71H
      MOV DL,AL
      MOV SI,DX   ;в SI контрольная сумма
;сосчитать
      MOV CX,30  ;количество ячеек
      MOV BL,10H  ;начальный адрес
      XOR DX,DX   ;будет накапливаться сумма
      XOR AH,AH
LOO:
      MOV   AL,BL
      OUT   70H,AL
      JMP   $+2
      IN    AL,71H
      ADD   DX,AX
      INC   BL
      LOOP  LOO
;сравниваем
      CMP   DX,SI
      JZ    YES
      LEA   DX,MES2
      JMP   SHORT _END
```

```
YES:
        LEA   DX,MES1
_END:
        MOV   AH,9
        INT   21H
        RET
;данные
MES1  DB 'Контрольная сумма CMOS в порядке.',13,10,'$'
MES2  DB 'Обнаружено несовпадение контрольной суммы.', 13,10,'$'
CODE  ENDS
        END BEGIN
```

Рис. 23.3. Проверка контрольной суммы CMOS.

V. Тест клавиатуры.

Ниже приводится программа, тестирующая клавиатуру. Программа полностью основана на справочном материале, содержащемся в Приложении 9. Предлагаемый тест нельзя назвать всеобъемлющим. Читатель легко может развить данную программу, используя данные Приложения 9. В этом приложении Вы найдете еще одну программу управления клавиатурой непосредственно через ее регистры.

```
CODE  SEGMENT
        ASSUME CS:CODE, DS:CODE
        ORG  100H
BEGIN:
        CLI
;ждем, когда можно послать команду
        CALL WAIT_OUT_BUF  ;свободен выходной буфер?
        CALL WAIT_IN_BUF   ;свободен входной буфер?
/команда получения управляющего байта
        MOV  AL,20H
        OUT  64H,AL
        CALL WAIT_IN_BUF   /команда принята?
        CALL WAIT_OUT      /управляющий байт в буфере?
        IN   AL,60H
/запомнить исходное состояние управляющего байта
        MOV  _AL,AL
/**********************
/ждем, когда можно послать команду
        CALL WAIT_OUT_BUF  ;свободен выходной буфер?
        CALL WAIT_IN_BUF   ;свободен входной буфер?
;внутренний тест контроллера
        MOV  AL,0AAH
        OUT  64H,AL
```

```
      CALL WAIT_OUT       ;ответ получен?
      IN   AL,60H
      MOV  _TEST_K,AL
;восстановить управляющий байт
      CALL WAIT_IN_BUF
      MOV  AL,60H
      OUT  64H,AL    ;команда отправки управляющего байта
      CALL WAIT_IN_BUF
      MOV  AL,_AL
      OUT  60H,AL    ;отправляем управляющий байт
      CMP  _TEST_K,55H
      JNZ  ERR1
;команда клавиатуры 'эхо'
      CALL WAIT_IN_BUF
      MOV  AL,0EEH
      OUT  60H,AL
      CALL WAIT_OUT
      IN   AL,60H
      CMP  AL,0EEH
      JNZ  ERR2
;внутренний тест клавиатуры (сброс)
      CALL WAIT_IN_BUF
      MOV  AL,0FFH
      OUT  60H,AL
      CALL WAIT_OUT
      IN   AL,60H
      CMP  AL,0FAH   ;команда сброса получена?
      JNZ  ERR2
      CALL WAIT_OUT
      IN   AL,60H
      CMP  AL,0AAH   ;тест прошел успешно?
      JNZ  ERR2
      JMP  _END
ERR1:
      STI
      POP AX
      LEA DX,MES_1
      MOV AH,9
      INT 21H
      MOV AH,4CH
      MOV AL,2
      INT 21H
ERR2:
      STI
      POP AX
      LEA DX,MES_2
```

```
        MOV AH,9
        INT  21H
        MOV AH,4CH
        MOV AL,3
        INT  21H
ERR:
        STI
        POP  AX
        LEA  DX,MES_0
        MOV AH,9
        INT 21H
        MOV AH,4CH
        MOVAL,1
        INT  21H
_END:
        STI
        LEA  DX,MES
        MOV AH,9
        INT  21H
        MOV AH,4CH
        MOVAL,0
        INT  21H
;ждать, когда освободится входной буфер
WAIT_IN_BUF PROC
        XOR CX,CX
T1:
        IN AL,64H
        TEST AL,2
        LOOPNZ  T1
        JNZ ERR
        RETN
WAIT_IN_BUF ENDP
;ждать, когда освободится выходной буфер
WAIT_OUT_BUF PROC
        XOR CX,CX
T2:
        IN AL,64H
        TEST AL,1
        LOOPNZ T2
        JNZ   ERR
        RETN
WAIT_OUT_BUF ENDP
;ждать, когда заполнится выходной буфер
WAIT_OUT PROC
T3:
        IN AL,64H
        TEST AL,1
```

```
      JZ   T3
      RETN
WAIT_OUT ENDP
_AL  DB  ?
_TEST_K DB ?
MES_0   DB  'Ошибка клавиатуры во время теста!',13,10,'$'
MES_1   DB  'Тест контроллера клавиатуры не прошел!',13,10,'$'
MES_2   DB  'Тест клавиатуры не прошел!',13,10,'$'
MES DB  'Тест клавиатуры прошел успешно!', 13,10, '$'
CODE ENDS
      END BEGIN
```

Рис. 23.4. Программа тестирования клавиатуры.

VI. Часы реального времени и системный таймер.

Во время запуска MS DOS происходит установка системного таймера по часам реального времени (на компьютерах XT часов реального времени не было). В дальнейшем системные часы ориентированы на системный таймер, который увеличивается по прерыванию 08Н. В идеале системные часы и часы реального времени должны идти одинаково. Однако это не всегда так. Ниже я привожу программу, позволяющую сделать простейшее сравнение того, как идут эти часы. Сравнение производится визуально на экране. Программа может работать сколь угодно долго, то есть сравнение может быть произведено за достаточно длинный промежуток.

```
.286
CODE  SEGMENT
      ASSUME CS:CODE, DS:CODE
      ORG 100H
BEGIN:
;ES - для вывода на экран
      MOV AX,0B800H
      MOV ES,AX
;чистить экран
      MOV CX,4000
      XOR SI,SI
_CLS:
      MOV ES:[SI],0700H
      ADD SI,2
      LOOP _CLS
      MOV CX,23
;вывести надпись TEXT1
      LEA SI,TEXT1
      MOV DI,2000-80
```

```
_TEX1:
        MOV  AL, [SI]
        MOV ES:[DI],AL
        INC  SI
        ADD DI,2
        LOOP  _TEX1
        MOV CX,23
        LEA  SI,TEXT2
        MOV DI,2320-80
_TEX2:
;вывести надпись  TEXT2
        MOV AL, [SI]
        MOV ES:[DI],AL
        INC  SI
        ADD DI,2
        LOOP _TEX2
LOOP1:
;читать показание счетчика
        MOV AH,0
        INT 1AH
;преобразовать в показание системных часов
        CALL _COUNT
;вывести показание системных часов
        MOV DI,2320
        CALL PRINT
;читать показания встроенных часов
        MOV AH,2
        INT 1AH
;преобразовать формат BCD
        CALL _BCD
;вывести показание встроенных часов
        MOV DI,2000
        CALL PRINT
;проверить, не нажата ли клавиша
        MOV AH, 1
        INT 16H
        JZ LOOP1
;очистить буфер клавиатуры
        MOV AH, 6
        MOV DL,OFFH
        INT 21H
; выйти в ОС
        RETN
```

```
;процедура печати времени
PRINT PROC
      MOV CX,8
      LEA SI,SHABL
LOO1:
      MOV AL,[SI]
      MOV ES:[DI],AL
      MOV BYTE PTR ES:[DI]+1,12
      INC SI
      ADD DI,2
      LOOP LO01
      RETN
PRINT ENDP
;преобразование числа в BCD-формате в шаблон для печати
;времени по часам реального времени
_BCD PROC
;часы
      MOV AL,CH
      SHR AL,4
      ADD AL,48
      AND CH,00001111B
      CMP CH,10
      JB  _OK1
      ADD AL,1
      SUB CH,10
_OK1:
      ADD CH,48
      MOV BYTE PTR SHABL,AL
      MOV BYTE PTR SHABL+1,CH
;минуты
      MOV AL,CL
      SHR AL,4
      ADD AL,48
      AND CL,00001111B
      CMP CL,10
      JB  _OK2
      ADD AL,1
      SUB CL,10
_OK2:
      ADD CL,48
      MOV BYTE PTR SHABL+3,AL
      MOV BYTE PTR SHABL+4,CL
;секунды
      MOV AL,DH
      SHR AL,4
```

```
        ADD  AL,48
        AND  DH,00001111B
        CMP  DH,10
        JB   _OK3
        ADD  AL,1
        SUB  DH,10
_OK3 :
        ADD  DH,48
        MOV  BYTE  PTR  SHABL+6,AL
        MOV  BYTE  PTR  SHABL+7,DH
        RETN
_BCD ENDP
;преобразование показателя счетчика в шаблон для
;печати времени по системным часам
_COUNT  PROC
        MOV  AX,DX
        MOV  DX,CX
        MOV  CX,65520
        DIV  CX          ;часы в AX
        PUSH  DX
        MOV  CL,10
        DIV  CL
        ADD  AL,48
        ADD  AH,48
        MOV  BYTE  PTR  SHABL,AL
        MOV  BYTE  PTR  SHABL+1,AH
        POP  AX
        XOR  DX,DX
        MOV  CX,1092
        DIV  CX          ; минуты в AX
        PUSH  DX
        MOV  CL,10
        DIV  CL
        ADD  AL,48
        ADD  AH,48
        MOV  BYTE  PTR  SHABL+3,AL
        MOV  BYTE  PTR  SHABL+4,AH
        POP  AX
        XOR  DX,DX
        MOV  CX,18
        DIV  CX          ;секунды
        MOV  CL,10
        DIV  CL
        ADD  AL,48
```

```
      ADD AH,48
      MOV BYTE PTR SHABL+6,AL
      MOV BYTE PTR SHABL+7,AH
      RETN
_COUNT ENDP
;шаблон
SHABL DB  48,48,':',48,48,':',48,48
TEXT1 DB  'Часы реального времени:'
TEXT2 DB  'Системные часы:         '
CODE ENDS
      END BEGIN
```

Рис. 23.5. Сравнение хода часов реального времени и системных часов.

Приведенная выше программа основывается на результатах вызова прерывания 1AH (см. Приложение 8). Советую Вам разобраться в процедурах преобразования BCD-формата и преобразования счетчика системного таймера.

Глава 24. Начала программирования для WINDOWS.

Отворил я окно...

А.К. Толстой

Окно - замечательное изобретение. Тысячелетия человечество наблюдало мир из окна. Его можно было закрывать и открывать, относительно него можно было двигаться. Окно на экране – это попытка создания виртуального мира. Возвращение к нашим истокам, если хотите.

Материал, излагаемый в данной главе, не является руководством по программированию в среде WINDOWS. Это слишком обширный вопрос. Здесь лишь обозначаются некоторые ключевые моменты, которые помогут Вам в программировании под WINDOWS на ассемблере. По этой же причине не приводится справочная информация по WINDOWS. Ее слишком много.

На первый взгляд кажется, что программировать на ассемблере под WINDOWS нет необходимости. В принципе, имея в руках такой мощный инструмент, как Си, ассемблер можно использовать для каких-либо вставок, чтобы оптимизировать программу. Хочу, однако, заметить, что программирование на ассемблере в среде WINDOWS весьма приближается к программированию на Си в связи с тем, что в тех и других программах приходится использовать вызов стандартных функций операционной системы (функции API - APLICATION PROGRAMM INTERFACE). Все общение с компьютером происходит через эти функции. Вы скоро увидите, что написание небольших программ для WINDOWS на ассемблере так же просто, как и на Си. В этой Главе Вам особенно понадобятся знания, полученные из Главы 15.

Читатель, наверное, заметил тенденцию последних лет - программные пакеты занимают все больше и больше места. Это весьма странное явление. Ведь программа, работающая под WINDOWS, наоборот должна становиться компактнее по сравнению с аналогичной программой для MS DOS. Действительно, WINDOWS предоставляет программисту огромное количество функций. Там, где в MS DOS приходилось писать свою процедуру, здесь достаточно вызвать функцию с подходящими параметрами. Так что же происходит? Как всегда, это одна из великих мистификаций. Человечество не может без этого обойтись. И у пользователя возникает представление, что чем больше пакет и чем больше в нем файлов, тем это более солидный продукт. Увы, не всегда так. Большие, громоздкие пакеты часто сами не справляются с собой как раз по причине огромного количества файлов. Причина появления программных монстров связана прежде всего с тем, что производство программ становится все более и более индустриальным, я бы сказал, конвейерным делом. А разве можно сделать пакет совершенным и компактным в этой ситуации. И вот уже индустрия программных продуктов начинает диктовать нам условия: "Вы должны иметь такое и такое оборудование, а иначе наше программное обеспечение не будет у вас работать". Каково? Но мы то с Вами не конвейерщики, слава Богу, мы учимся делать штучный товар. Итак, вперед!

I. Некоторые общие положения.

Вначале сделаем ряд кратких замечаний, которые, на мой взгляд, являются ключевыми. В дальнейшем мы разберем их более подробно.

1. Программирование в среде WINDOWS основывается на использовании функций API (APPLICATION PROGRAM INTERFACE). Количество таких функций составляет несколько сотен. Ваша программа будет состоять из большого количества вызовов этих функций.
2. Список функций API и их описание проще всего брать из документации по языку Си. Ниже будет подробно рассказано, как использовать это описание в языке ассемблера.
3. Главным элементом (объектом) программы в среде WINDOWS является 'Окно'. Для каждого окна определяется процедура прерывания. Эта процедура 'вылавливает' всю информацию, предназначенную этому окну.
4. В каждом окне можно открывать другие объекты (кнопки, полосы прокрутки и др.). Информация к этим объектам (и от них) поступает через процедуру окна. Кстати, все эти объекты, по сути, тоже являются окнами, но имеющими специальные свойства.
5. Для обычных прикладных программ нет практически никакой разницы в программировании для WINDOWS 95-98 и WINDOWS 3.1. Основное различие между этими системами для программистов заключается в следующих моментах:
   - WINDOWS 95 поддерживает 32-битную адресацию, в связи с этим следует иметь в виду, что некоторые входные параметры API-функций изменили свою разрядность,
   - WINDOWS 95 поддерживает многозадачность с автовыгрузкой, такую многозадачность иногда называют истинной,
   - WINDOWS 95 поддерживает многозадачность не только на уровне процессов (программ), но и на уровне потоков (частей программ), отдельную процедуру программы можно объявить потоком (Thread), и она станет работать как независимая программа,
   - в WINDOWS 95 каждая задача имеет свою входную очередь сообщений, тогда как в предыдущих версиях, был один входной поток,
   - WINDOWS 95 предоставляет в распоряжение программистам специальные окна для текстовой информации, они называются консоли, работа с которыми несколько напоминает работу с MS DOS,
   - WINDOWS 95 рассматривает адресное пространство как линейное, WINDOWS 3.1 работала в сегментированном адресном пространстве,
   - WINDOWS 95 предоставляет новые стандартные элементы, объекты управления и большое количество новых функций.

Даже когда мы перейдем с Вами к 32-битному программированию (глава 25), сама структура программы не претерпит практически никаких изменений.

II. Вызов стандартных API-функций.

Программируя в MS DOS, мы могли использовать различные прерывания либо обращаться непосредственно к оборудованию через порты ввода-вывода. Это созда-

вало значительные проблемы, т.к. оборудование на разных компьютерах могло отличаться друг от друга. При программировании же в WINDOWS для Вас будут существовать только API-функции. Все сношение с внешним миром, то есть с компьютером, должно производиться через эти функции.

Как правило, в поставке любого языка программирования, который предполагается использовать для программирования под WINDOWS, имеется специальная библиотека, которая позволяет вызывать эти функции. Имеется такая библиотека и в макроассемблере: LIBW.LIB. В Главе 14 дается структура загружаемых модулей (ЕХЕ-программ), которые работают в среде WINDOWS. Может сразу возникнуть вопрос, каким образом этот модуль может быть создан. Все очень просто, никаких специальных ключей не требуется. Если в Вашей программе вызывается хотя бы одна функция из библиотеки LIBW и эта библиотека будет указана программе LINK, то компоновщик создаст модуль для WINDOWS. Ниже приводится пример простой программы (см. Рис. 24.1), которая ничего не делает. Пример приводится для демонстрации самого процесса трансляции. Процедура INITTASK является библиотечной, но в данной программе она вызывается формально. В дальнейшем DOSSEG, DGOUP, INITTASK найдут свое разъяснение. Все последующие программы будут иметь точно такую же структуру. Нашей дальнейшей задачей будет наполнение этой структуры тем или иным содержанием. Замечу также, что для трансляции программ в этой главе используется макроассемблер фирмы MICROSOFT версии 6.1 и компоновщик версии 5.0. Последовательность команд трансляции следующая:

```
ML -c PROG.ASM; 65
LINK PROG.OBJ
```

На вопрос о библиотеке ответьте LIBW. Можно написать также строку:

```
LINK PROG,PROG,,LIBW;
```

Общий вид командной строки для LINK.EXE версии 5.0:

```
LINK OBJFILE,EXEFILE,MAPFILE,LIBFILE,DEFFILE;
```

А вот и сама программа.

```
.DOSSEG
DGROUP GROUP    DATA,  STA
      ASSUME CS:CODE, DS:DGROUP
EXTRN  INITTASK:FAR
EXTRN  DOS3CALL:FAR
;сегмент стека
STA SEGMENT STACK 'STACK'
      DW 2000 DUP(?)
STA ENDS
```

[65] Чтобы макроассемблер правильно работал под Windows, Вам придется в файл system.ini в раздел [386Enh] поместить строку типа device=\путь\DOSXNT.386.

17 – 4072

```
;сегмент данных
DATA SEGMENT WORD 'DATA'
DATA ENDS
;сегмент кода
CODE SEGMENT WORD  'CODE'
_BEGIN:
;ничего не значащий вызов
      PUSH AX
      CALL INITTASK
;выход из программы
      MOV  AH,4CH
      CALL DOS3CALL   ;вызов подпрограмм INT 21H
CODE ENDS
      END  _BEGIN
```

Рис. 24.1. Пример простой программы.

Еще раз подчеркну, что пока мы занимаемся 16-битными программами под Windows. Эти программы будут работать как под Windows 95-98, так и под Windows 3.1, о которой многие уже, наверное, забыли.

Чтобы использовать функции API в своей программе, предлагается несколько положений, которые помогут Вам это делать.

1. Функции API хорошо описаны в документации к языкам Си (проще использовать обычный HELPER). В справочниках по функциям API обычно также используется Си-нотация.
2. Следует в первую очередь обратить внимание на тип, количество и порядок посылаемых в функцию параметров. Вам "вручную" придется посылать эти параметры. В этой главе Вы встретите прямой (слева направо) и обратный (справа налево) порядок посылки параметров.
3. Функция, как правило, что-то возвращает. Следует четко представлять, где возвращается значение функции (в каком регистре или регистрах).
4. Из главы 15 мы знаем, что в том случае, когда вызываемая функция получает параметры, после возврата из нее стек должен быть освобожден. Вызов функций API происходит согласно правилам языка Паскаль, поэтому освобождать стек после возврата из функций API не нужно.

Рассмотрим теперь несколько конкретных функций. Причем нас пока не будет интересовать, что делают эти функции. Интересен вопрос: как вызвать эти функции в программе на языке ассемблера. Напоминаю читателю, что в Си принято различать прописные и строчные буквы. Мы традиционно не будем придерживаться этого (где это будет необходимо, будет отмечено особо). Итак:

```
BOOL GETMESSAGE(LPMSG, HWND, UMSGFILTERMIN, UMSGFILTERMAX)
```

Данная функция имеет тип BOOL. Однако мы понимаем, что логическая переменная - это фикция. Речь идет о целой переменной. Просто принимается соглашение, что значение 0 - ложь, а не ноль (обычно 1) - истина. А мы знаем, что обычная целая

величина (или слово со знаком) возвращается в регистре AX. С возвращаемой величиной мы разобрались. Смотрим на типы параметров:

LPMSG - дальний указатель на структуру MSG (сообщение системной очереди),

HWND - тип WORD,
UMSGFILTERMIN - тип WORD,
UMSGFILTERMAX - тип WORD.

Смысл данной функции будет понятен в дальнейшем. С формальной же точки зрения мы уже готовы поместить ее вызов в своей программе. Пусть три последних параметра равны нулю, а структура для сообщения системной очереди имеет имя MES. Тогда имеем:

```
PUSH  DS          ; поместить
LEA AX,MES        ;в стек
PUSH AX           ;дальний указатель
PUSH O
PUSH O
PUSH O
CALL  GETMESSAGE
CMP AX,0  ;что возвращает?
JZ _EXIT
```

Вот и все. Следует отметить, что хотя мы говорим здесь о функции СИ, но посылка параметров осуществляется так же, как на Паскале, т.е. слева направо. Вот еще пример:

```
LONG  DISPATCHMESSAGE(LPMSG);
CONST  MSG  FAR*  LPMSG;
```

Функция формирует сообщение для окна из очереди сообщений для данного приложения (см. ниже). Возвращаемое значение имеет тип LONG. Напомню, что возвращается такое значение в паре регистров DX:AX (Глава 15). Впрочем, это значение нам не понадобится. Параметр, который посылается в функцию, есть указатель на некоторую структуру MSG, о которой будет сказано ниже. Вот вызов указанной функции:

```
PUSH  DS
LEA  AX,MSG
PUSH  AX
CALL     DISPATCHMESSAGE
```

Подведем теперь некоторый итог.

1. Параметры передаются через стек. Порядок передачи параметров - слева направо.
2. При передаче параметров длиной больше, чем слово, вначале передается старшее слово, а затем младшее.

Примеров с вызовом функций, я думаю, уже достаточно, рассмотрим теперь некоторые необходимые структуры. Самая важная в WINDOWS структура - это структура сообщения системной очереди. Ниже она имеет имя MESSA.

```
MESSA       STRUCT
HWND         DW      ?
MESSAGE      DW      ?
WPARAM       DW      ?
LPARAM       DD      ?
TIME         DD      ?
X            DW      ?
Y            DW      ?
MESSA       ENDS
```

Напомню (Приложение 3), что мы, таким образом, создали шаблон структуры. Для того чтобы зарезервировать место для такой структуры следует указать директиву:

```
MSG   MESSA   O.
```

На Си подобная структура имеет следующий вид.

```
TYPEDEF STRUCT TAGMSG {        /* MSG */
        HWND     HWND;
        UINT     MESSAGE;
        WPARAM   WPARAM;
        LPARAM   LPARAM;
        DWORD    TIME;
        POINT    PT;
}  MSG;
```

PT - структура, состоящая из координат X, Y. Вдумайтесь теперь, для чего вводятся новые типы HWND, UINT, WOARAM, если все это WORD. Если честно, то я не понимаю для чего, и это является дополнительным стимулом в моей любви к ассемблеру.

Еще одна важная структура определяет класс окна.

```
WNDCLASS              STRUCT
STYLE            DW ?
LPFNWNDPROC      DD ?   ;указатель на процедуру обработки
CBCLSEXTRA       DW ?
CBWNDEXTRA       DW ?
HINSTANCE        DW ?
HICON            DW ?
HCURSOR          DW ?
```

```
HBRBACKGROUND    DW ?
LPSZMENUNAME     DD ?   ;указатель на строку
LPSZCLASSNAME    DD ?   ;указатель на строку
WNDCLASS         ENDS
```

В данной структуре хранятся параметры класса окна. На Си эта структура выглядит так:

```
TYPEDEF STRUCT TAGWNDCLASS {        /* WC */
      UINT        STYLE;
      WNDPROC     LPFNWNDPROC;
      INT         CBCLSEXTRA;
      INT         CBWNDEXTRA;
      HINSTANCE   HINSTANCE;
      HICON       HICON;
      HCURSOR     HCURSOR;
      HBRUSH      HBRBACKGROUND;
      LPCSTR      LPSZMENUNAME;
      LPCSTR      LPSZCLASSNAME;
} WNDCLASS;
```

III. Структура программы.

В нашей главе мы будем рассматривать в основном программы, состоящие из одного сегмента кода и одного сегмента данных. Принято называть такую модель памяти SMALL - маленькой. При программировании на ассемблере ее вполне достаточно. На Рис. 24.2 изображена структура такой программы.

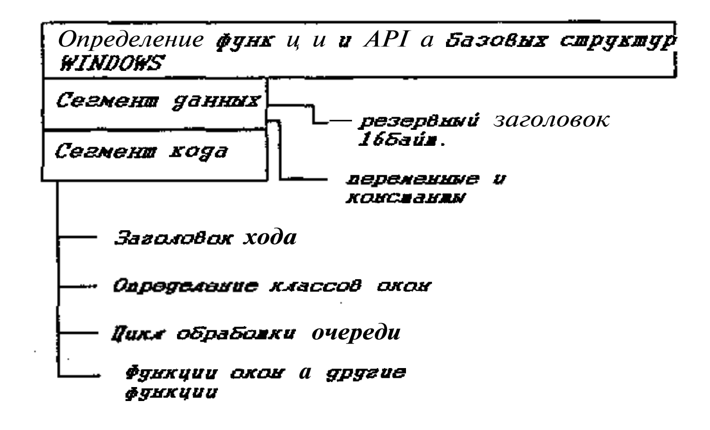


Рис. 24.2. Структура программы.

Стоит более подробно рассмотреть сегмент кода.

1. Заголовок кода. В заголовке кода присутствует вызов трех функций API. Интересно, что при создании программы на Си данный заголовок создается автоматически, и программист может и не подозревать о существовании этих функций. Мы же стараемся понять все. Вот эти функции:

INITTASK - инициализирует регистры, командную строку и память. Входных параметров не требует. Вызывается первой. Возвращаемые значения регистров: AX = 1 (O - ошибка), CX - размер стека, DI - уникальный номер для данной задачи, DX - параметр NCMDSHOW (см. ниже), ES - сегментный адрес (селектор) PSP, ES:BX - адрес командной строки, SI - уникальный номер для ранее запущенного того же приложения. В WINDOWS 3.1 при запуске приложения несколько раз, каждый раз в память загружается только часть сегментов, часть сегментов является общим ресурсом. Этим достигалась экономия памяти. В WINDOWS 95 от этого отказались. Каждая запущенная задача является изолированной и независимой. В WINDOWS 95 SI всегда будет содержать 0. Кроме упомянутого, данная процедура заполняет резервный заголовок сегмента данных.

INITAPP - инициализирует очередь событий для данного приложения. Вызов:

```
PUSH  DI          ;уникальный номер задачи
CALL  INITAPP
```

В случае ошибки данная функция возвращает 0, иначе не нулевое значение.

WAITEVENT - проверяет наличие событий для указанного приложения. Если событие есть, то оно удаляется из очереди.

```
PUSH AX          ;AX - номер приложения, если 0 то текущее.
CALL WAITEVENT
```

Итак, в начале сегмента кода имеем следующий фрагмент.

```
CALL  INITTASK    '            ;инициализировать задачу
OR AX,AX                       ;CX - границы стека
JZ  TO_OS                      ;
MOV  HPREV,SI                  ;номер предыдущего прил.
MOV  HINST,DI                  ;номер для новой задачи
MOV  WORD  PTR  LPSZCMD,BX     ;ES:BX - адрес командной строки
MOV  WORD  PTR  LPSZCMD+2,ES   ;
MOV  CMDSHOW,DX                ;экранный параметр
PUSH  0                        ;текущая задача
CALL  WAITEVENT                ;очистить очередь событий
PUSH  HINST
CALL  INITAPP                  ;инициализировать приложения
OR  AX,AX
JZ  TO_OS
```

```
 CALL  MAIN                        ;запуск основной части
_TO_OS:
 MOV AH,4CH
 CALL DOS3CALL                      ; выйти из программы
```

Естественно, все используемые здесь функции должны быть описаны как внешние. В начало программы нужно вставить следующие строки.

```
EXTRN INITTASK:FAR
EXTRN INITAPP:FAR
EXTRN WAITEVENT:FAR
EXTRN DOS3CALL:FAR
```

2. Определение классов окон. В Вашей программе обязательно **будет,** по крайней мере, одно окно. Любое конкретное окно принадлежит какому-либо классу. Класс сначала должен быть зарегистрирован. После регистрации можно создавать любое количество экземпляров окон данного класса. Есть уже определенные классы - такие окна можно создавать сразу. Итак, ниже приводится фрагмент, который регистрирует класс окна. Заметим, однако, что задача может не иметь окон вовсе. Такая задача не будет представлена на нижней панели окна. Однако список задач, который можно получить, например, по **CTRL+ALT+DEL,** будет содержать **и** эту задачу.

```
        LEA   DI,WNDCLASS
;стиль окна в STYLE
        MOV       AX, STYLE
        MOV       WNDCLASS.STYLE, AX ;значения стилей см.ниже
;процедура обработки
        MOV       BX,OFFSET WNDPROC
        MOV       WORD PTR WNDCLASS.LPFNWNDPROC, BX
        MOV        BX,SEG WNDPROC
        MOV       WORD PTR WNDCLASS.LPFNWNDPROC+2, BX
;----------------------------------
        XOR       AX,AX
;резервные байты в конце резервируемой структуры
        MOV       WNDCLASS.CBCLSEXTRA, AX
;резервные байты в конце структуры для каждого окна
        MOV       WNDCLASS.CBWNDEXTRA, AX
;иконка окна отсутствует
        MOV       WNDCLASS.HICON, AX
;номер запускаемой задачи
        MOV       AX,HINST
        MOV       WNDCLASS.HINSTANCE, AX
;определить номер стандартного курсора
        PUSH      O
```

```
        PUSH    DS
        PUSH    CURSOR
        CALL    LOADCURSOR
        MOV     WNDCLASS.HCURSOR, AX
;определить номер стандартного объекта
        PUSH    0           ;WHITE_BRUSH
        CALL    GETSTOCKOBJECT
;цвет фона
        MOV     WNDCLASS.HBRBACKGROUND, AX
;имя меню из файла ресурсов (отсутствует = NULL)
        XOR     AX, AX
        MOV     WORD PTR WNDCLASS.LPSZMENUNAME,AX
        MOV     WORD PTR WNDCLASS.LPSZMENUNAME+2,AX
;указатель на строку, содержащую имя класса
        MOV     BX,OFFSET CLAS_NAME
        MOV     WORD PTR WNDCLASS.LPSZCLASSNAME,BX
        MOV     WORD PTR WNDCLASS.LPSZCLASSNAME+2,DS
;вызов процедуры регистрации
        PUSH DS     ;указатель на
        PUSH DI     ;структуры WNDCLASS
        CALL  REGISTERCLASS
```

После того как класс был зарегистрирован, структура WNDCLASS более не нужна. Так что эта структура может создаваться в блоке памяти, который потом можно освободить. После того как все классы зарегистрированы, обычно создают главное окно задачи, остальные окна создаются по мере необходимости. Ниже приведена процедура создания окна.

API предлагает две функции создания окон— CREATEWINDOW и CREATEWINDOWEX. Вторая функция отличается от первой только наличием параметра (двойное слово) расширенного стиля окна. Поэтому мы рассмотрим только первую функцию. В Си-нотации она имеет вид:

```
HWND CREATEWINDOWEX(DWEXSTYLE, LPSZCLASSNAME, LPSZWINDOWNAME,
DWSTYLE, X, Y, NWIDTH, NHEIGHT, HWNDPARENT, HMENU, HINST,
LPVCREATEPARAMS)
```

Не надо пугаться вида этой функции, впрочем, венгерская нотация любого сведет с ума. Вот вызов этой функции на ассемблере:

```
;параметр расширенного стиля окна
        MOV     BX,HIGHWORD EX_STYLE
        PUSH BX
        MOV     BX,LOWWORD EX_STYLE
        PUSH BX
```

```
;адрес строки-имени класса окна
        PUSH  DS
        LEA  BX,RNAME
        PUSH  BX
;адрес строки-заголовка окна
        PUSH  DS
        LEA  BX,SZAPPNAME
        PUSH  BX
;стиль окна
        MOV  BX,HIGHWORD STYLE
        PUSH BX
        MOV   BX,LOWWORD STYLE
        PUSH BX
;координата X левого верхнего угла
        PUSH XSTART
;координата Y левого верхнего угла
        PUSH YSTART
;ширина окна
        PUSH  DXCLIENT
;высота окна
        PUSH DYCLIENT
;номер окна-родителя
        PUSH 0
;номер (идентификатор) меню окна
        PUSH 0
;номер задачи
        PUSH HINST
;адрес блока параметров окна (нет)
        PUSH 0
        PUSH 0
        CALL   CREATEWINDOWEX
```

Как видите, ничего сложного, и можно перейти к следующей части кода. Но прежде замечу, что Вы можете в своей программе использовать вызов обычных прерываний, как в MS DOS. Дело в том, что WINDOWS поддерживает стандартные DOS-овские прерывания. Скажем, вместо CALL DOS3CALL можно просто ставить INT 21H. Особенно полезными будут функции работы с файлами. Конечно, надо иметь в виду, какие функции (и прерывания) имеют смысл для WINDOWS, а какие нет.

3. Цикл обработки очереди. Данный цикл играет роль диспетчера, который захватывает получаемые приложением сообщения и направляет их окнам.

```
                LOOP1:
;извлечение сообщения из очереди
                        PUSH  DS
                        LEA BX,MSG   ;указатель на структуру
                        PUSH BX      ;сообщения
```

```
                PUSH  0
                PUSH  0
                PUSH  0
                CALL  GETMESSAGE
;проверка - не получено сообщение "выход"
                CMP AX,0
                JZ    NO_LOOP1
;перевод всех пришедших сообщений к стандарту ANSI
                PUSH  DS
                LEA BX,MSG
                PUSH  BX
                CALL  TRANSLATEMESSAGE
;указать WINDOWS передать данное сообщение соответствующему
окну
                PUSH  DS
                LEA BX,MSG
                PUSH  BX
                CALL   DISPATCHMESSAGE
;замкнуть цикл  (петлю)
                JMP  SHORT  LOOP1
```

4. Данный раздел назван "функции окон и другие функции". С другими функциями понятно. Рассмотри более подробно структуру функции окна. Фактически эти функции являются процедурами прерывания, которые обрабатывают сообщения, пришедшие данному окну как посланные из цикла обработки очереди, так и непосредственно операционной системой. Здесь, как и раньше, нам придется перевести язык Си на ассемблер. Опираясь на сведения данной главы и на главы 11,13,15, мы сделаем это без труда. Итак, заголовок функции окна на Си можно представить в виде:

```
VOID FUN_WIND(UNSIGNED INT HWND, UNSIGNED INT MES, INT WPARAM,
LONG LPARAM).
```

Не представляет никакого труда написать структуру такой функции на языке ассемблера:

```
FUN_WIND PROC
        PUSH  BP
        MOV  BP,SP
        .
        MOV AX,[BP+0EH]    ;HWND - номер приложения
        MOVAX,[BP+0CH]     ;MES - номер сообщения
        MOV AX,[BP+0AH]    ;WPARAM - параметр сообщения
        MOV AX,[BP+8]      ;старшее слово параметра LPARAM
        MOVAX,[BP+6]       ;младшее слово параметра LPARAM
        .
```

```
        POP  BP
        RET 10              ;освобождаем стек
FUN_WIND ENDP
```

Как видите, все довольно просто. И мы практически готовы написать простую программу для работы в WINDOWS. Конкретные значения параметров, констант, как и раньше, будем узнавать в процессе программирования.

IV. Пример программы.

Изложенный выше материал позволяет написать простую программу (Рис. 23.4). Программа открывает стандартное окно с обычными атрибутами управления: закрытие, минимизация, распахивание, перемещение и изменение размеров. Константы (стиль класса, стиль окна, вид курсора и т.д.) можно взять в каком-нибудь из файлов: WINDOWS.H, WIN.INC или прямо из системы помощи по Си, что мы и сделали. Например, стиль окна взят в виде: OOOCFOOOOH.

Эта константа означает следующее:

OOOCFOOOOH - WS_OVERLAPPEDWINDOW задает тип окна:

1) - с заголовком и бордюром,

2) - с полосой для заголовка,

3) - с управляющими элементами (правый верхний угол),

4) - с толстой окаемкой вокруг окна,

5) - с кнопками минимизации и максимизации.

Используя помощь к Borland Си, Вы можете выяснить смысл и значение других констант. Обращу Ваше внимание на тот факт, что каждому свойству в 4-байтной константе соответствует свой бит. Мы намеренно ставим директиву .286, чтобы подчеркнуть, что пока работаем в 16-битном режиме.

```
.286
.DOSSEG
DGROUP GROUP    DATA,  STA
                 ASSUME CS:CODE, DS:DGROUP
;внешние процедуры
        EXTRN  INITTASK:FAR
        EXTRN  INITAPP:FAR
        EXTRN  WAITEVENT:FAR
        EXTRN  DOS3CALL:FAR
        EXTRN  REGISTERCLASS:FAR
        EXTRN  LOADCURSOR:FAR
        EXTRN  GETSTOCKOBJECT:FAR
        EXTRN  GETMESSAGE:FAR
        EXTRN  TRANSLATEMESSAGE:FAR
        EXTRN  DISPATCHMESSAGE:FAR
        EXTRN  CREATEWINDOW:FAR
```

```
       EXTRN  CREATEWINDOWEX:FAR
       EXTRN  UPDATEWINDOW:FAR
       EXTRN  SHOWWINDOW:FAR
       EXTRN  POSTQUITMESSAGE:FAR
       EXTRN  DEFWINDOWPROC:FAR
;шаблоны
WNDCL                STRUCT
STYLE              DW 0   ;стиль класса окна
LPFNWNDPROC        DD 0   /указатель на процедуру обработки
CBCLSEXTRA         DW O
CBWNDEXTRA         DW O
HINSTANCE          DW O
HICON              DW O
HCURSOR            DW O
HBRBACKGROUND      DW O
LPSZMENUNAME       DD O   ;указатель на строку
LPSZCLASSNAME      DD O   ;указатель на строку
WNDCL          ENDS
MESSA            STRUCT
HWND               DW  ?
MESSAGE            DW  ?
WPARAM             DW  ?
LPARAM             DD  ?
TIME               DW  ?
X                  DW  ?
Y                  DW  ?
MESSA             ENDS
;сегмент стека
STA SEGMENT STACK 'STACK'
       DW 2000 DUP(?)
STA ENDS
;сегмент данных
DATA SEGMENT WORD 'DATA'
;вначале 16 байт - резерв, необходимый 16-битному приложению
/для правильной работы в среде Windows
        DWORD  O
        WORD    5
        WORD    5 DUP (0)
HPREV    DW ?
HINST    DW ?
LPSZCMD DD ?
CMDSHOW DW ?
;структура для создания класса
WNDCLASS WNDCL <>
```

```
;структура сообщения
MSG MESSA O
/имя класса окна
CLAS_NAME DB 'PRIVET',0
;заголовок окна
APP_NAME DB 'FIRST PROGRAM.', 0
;тип курсора
CURSOR EQU 00007F00H
;стиль окна
STYLE EQU 000CF0000H
;параметры окна
XSTART DW  100
YSTART DW  100
DXCLIENT DW 300
DYCLIENT DW 200
DATA ENDS
;сегмент кода
CODE SEGMENT WORD  'CODE'
_BEGIN:
;I. Начальный код
      CALL INITTASK              ;инициализировать задачу
      OR AX,AX                   ;CX - границы стека
      JZ _ERR                    ;
      MOV HPREV,SI               ;номер предыдущего прил.
      MOV HINST,DI               ;номер для новой задачи
      MOV WORD PTR LPSZCMD,BX    ;ES:BX - адрес командной
строки
      MOV WORD PTR LPSZCMD+2,ES  ;
      MOV CMDSHOW,DX             ;экранный параметр
      PUSH 0                     ;текущая задача
      CALL WAITEVENT             ;очистить очередь событий
      PUSH HINST
      CALL INITAPP               ;инициализировать приложения
      OR AX,AX
      JZ _ERR
      CALL MAIN                  ;запуск основной части
_TO_OS:
      MOV AH,4CH
      CALL DOS3CALL              ; выйти из программы
_ERR:
      CALL BEEP
      JMP SHORT _TO_OS
;основная процедура
;*****************************************************
```

```
MAIN PROC
/II. Регистрация класса окна
;стиль окна NULL - стандартное окно
      MOV       WNDCLASS.STYLE,0
;процедура обработки
      LEA       BX,WNDPROC
      MOV       WORD PTR WNDCLASS.LPFNWNDPROC, BX
      MOV       BX,CS
      MOV       WORD PTR WNDCLASS.LPFNWNDPROC+2,BX
;-------------------------------
;резервные байты в конце резервируемой структуры
      MOV       WNDCLASS.CBCLSEXTRA, O
;резервные байты в конце структуры для каждого окна
      MOV       WNDCLASS.CBWNDEXTRA, O
;иконка окна отсутствует
      MOV       WNDCLAS S.HICON, O
;номер запускаемой задачи
      MOV       AX,HINST
      MOV       WNDCLASS.HINSTANCE,AX
;определить номер стандартного курсора
      PUSH      O
      PUSH      DS
      PUSH      CURSOR
      CALL      LOADCURSOR
      MOV       WNDCLASS.HCURSOR, AX
/определить номер стандартного объекта
      PUSH      0           ;WHITE_BRUSH
      CALL      GETSTOCKOBJECT
;цвет фона
      MOV       WNDCLASS.HBRBACKGROUND, AX
/имя меню из файла ресурсов (отсутствует = NULL)
      MOV       WORD PTR WNDCLASS.LPSZMENUNAME, O
      MOV       WORD PTR WNDCLASS.LPSZMENUNAME+2, 0
/указатель на строку, содержащую имя класса
      LEA       BX,CLAS_NAME
      MOV       WORD PTR WNDCLASS.LPSZCLASSNAME, BX
      MOV       WORD PTR WNDCLASS.LPSZCLASSNAME+2, DS
/вызов процедуры регистрации
      PUSH DS    /указатель на
      LEA DI,WNDCLASS
      PUSH DI    /структуры WNDCLASS
      CALL REGISTERCLASS
      CMP AX,0
      JNZ _OK1
```

```
;звуковой сигнал при ошибке
      CALL BEEP
      RET            ;ошибка при регистрации
_OK1:
;III. Создание окна
;адрес строки-имени класса окна
      PUSH DS
      LEA BX,CLAS_NAME
      PUSH BX
;адрес строки-заголовка окна
      PUSH DS
      LEA BX,APP_NAME
      PUSH BX
;стиль окна
      MOV BX,HIGHWORD STYLE
      PUSH BX
      MOV  BX,LOWWORD STYLE
      PUSH BX
;координата X левого верхнего угла
      PUSH XSTART
;координата Y левого верхнего угла
      PUSH YSTART
;ширина окна
      PUSH DXCLIENT
;высота окна
      PUSH DYCLIENT
;номер окна-родителя
      PUSH 0
;номер (идентификатор) меню окна
      PUSH 0 ;NULL
;номер задачи
      PUSH HINST
;адрес блока параметров окна (нет)
      PUSH 0
      PUSH 0
      CALL  CREATEWINDOW
      CMP AX,0
      JNZ NO_NULL
      CALL BEEP
      RET         ;ошибка при создании окна
```

```
;установка для окна состояния видимости (окно или
;пиктограмма) согласно параметру CMDSHOW и его отображение
NO_NULL:
      MOV SI,AX
      PUSH SI
      PUSH CMDSHOW
      CALL SHOWWINDOW
;посылка команды обновления области окна (команда WM_PAINT)
;сообщение посылается непосредственно окну
      PUSH SI
      CALL UPDATEWINDOW
;IV. Цикл ожидания
      LOOP1:
;извлечение сообщения из очереди
      PUSH DS
      LEA BX,MSG   ;указатель на структуру
      PUSH BX      ; сообщения
      PUSH 0
      PUSH 0
      PUSH 0
      CALL GETMESSAGE
;проверка - не получено сообщение "выход"
      CMP AX,0
      JZ  NO_LOOP1
;перевод всех пришедших сообщений к стандарту ANSI
      PUSH DS
      LEA BX,MSG
      PUSH BX
      CALL TRANSLATEMESSAGE
;указать WINDOWS передать данное сообщение соответствующему
окну
      PUSH DS
      LEA BX,MSG
      PUSH BX
      CALL DISPATCHMESSAGE
;замкнуть цикл (петлю)
      JMP SHORT LOOP1
NO_LOOP1:
      RET
MAIN ENDP
;процедура для заданного класса окон
/WINDOWS передает в эту процедуру параметры:
;HWND - дескриптор (номер) окна, тип WORD
;MES - номер сообщения, тип WORD
```

```
;WPARAM - дополнительная информация о сообщении, тип WORD
;LPARAM - дополнительная информация о сообщении, тип DWORD
WNDPROC  PROC
      PUSH BP
      MOV  BP,SP
      MOV AX, [BP+OCH]    ;MES - номер сообщения
      CMP AX, 2  ;не сообщение ли о закрытии (2-сообщение о
закрытии)
      JNZ NEXT
;передать сообщение о закрытии приложения, это сообщение
;будет принято в цикле ожидания, и т.о. приложение завершит
свой путь
      PUSH O
      CALL  POSTQUITMESSAGE
      JMP _QUIT
NEXT:
;передать сообщение дальше WINDOWS
;своего рода правило вежливости - то, что не обработано
;процедурой обработки, предоставляется для обработки
;WINDOWS
      PUSH  [BP+OEH]        ;HWND
      PUSH  [BP+OCH]        ;MES - номер сообщения
      PUSH  [BP+0AH]        ;WPARAM
      PUSH  [BP+8]          ;HIGHWORD LPARAM
      PUSH  [BP+6]          ;LOWWORD LPARAM
      CALL  DEFWINDOWPROC
;*****************************************************
_QUIT:
      POP  BP
;вызов процедуры окна всегда дальний, поэтому RETF
      RETF  10              ;освобождаем стек от параметров
WNDPROC ENDP
;звуковой сигнал
BEEP  PROC
      MOV AH, 2
      MOV  DL,7
      CALL DOS3CALL  ;INT 21H
      RET
BEEP ENDP
CODE ENDS
      END   _BEGIN
```

Рис. 24.3. Простая программа, открывающая окно.

V. Простейшие органы управления.

Органы управления располагаются в текущем окне и относятся к предопределенным классам. Следовательно, органы управления регистрировать не надо - их достаточно создать. И это хорошо! Органы управления также могут (и должны) получать сообщения (например, щелчок мышью по кнопке). Замечательно, однако, то, что для них не нужно создавать свои процедуры обработки. Сообщение будет приходить в процедуру окна, где расположен орган управления, и должно распознаваться этой процедурой. Отличить же один орган от другого можно, анализируя значения параметров LPARAM и WPARAM. Видите, как замечательно! Одним "движением руки" вы создаете на экране кнопку, еще одно движение - и кнопка начинает работать. Ну что ж, приступим.

```
.286
.DOSSEG
DGROUP  GROUP
DATA,   STA
        ASSUME CS:CODE, DS:DGROUP
;внешние процедуры
EXTRN INITTASK:FAR
EXTRN INITAPP:FAR
EXTRN WAITEVENT:FAR
EXTRN DOS3CALL:FAR
EXTRN REGISTERCLASS:FAR
EXTRN LOADCURSOR:FAR
EXTRN GETSTOCKOBJECT:FAR
EXTRN GETMESSAGE:FAR
EXTRN TRANSLATEMESSAGE:FAR
EXTRN DISPATCHMESSAGE:FAR
EXTRN CREATEWINDOW:FAR
EXTRN CREATEWINDOWEX:FAR
EXTRN UPDATEWINDOW:FAR
EXTRN SHOWWINDOW:FAR
EXTRN POSTQUITMESSAGE:FAR
EXTRN DEFWINDOWPROC:FAR
;шаблоны
WNDCL              STRUCT
STYLE            DW 0   ;стиль класса окна
LPFNWNDPROC      DD 0   ;указатель на процедуру обработки
CBCLSEXTRA       DW 0
CBWNDEXTRA       DW 0
HINSTANCE        DW 0
HICON            DW 0
HCURSOR          DW 0
HBRBACKGROUND    DW 0
LPSZMENUNAME     DD 0   ;указатель на строку
LPSZCLASSNAME    DD 0   ;указатель на строку
```

```
WNDCL          ENDS
MESSA            STRUCT
HWND              DW  7
MESSAGE           DW  7
WPARAM            DW  7
LPARAM            DD  ?
TIME              DW  7
X                 DW  7
Y               .  DW  7
MESSA            ENDS
;сегмент стека
STA SEGMENT STACK 'STACK'
      DW 2000 DUP(?)
STA ENDS
;сегмент данных
DATA SEGMENT WORD 'DATA'
;16 байт - резерв
      DWORD  0
      WORD   5
      WORD   5 DUP (0)
HPREV   DW ?
HINST   DW ?
LPSZCMD DD ?
CMDSHOW DW ?
;структура для создания класса
WNDCLASS WNDCL <>
; структура сообщения
MSG MESSA O
;имя класса основного окна
CLAS_NAME DB 'PRIVET',0
;имя предопределенного класса
PRED_CLAS DB 'BUTTON',O
; заголовок окна
APP_NAME DB 0  ;нет
;надпись на кнопке
BUT_NAME DB 'QUIT',0
; тип курсора
CURSOR EQU 00007FOOH
;стиль окна
STYLE EQU  080000000H OR 000400000H
;стиль кнопки
STYLE_BUT EQU 040000000H OR 010000000H
; параметры окна
XSTART DW  100
YSTART DW  100
DXCLIENT DW 300
```

```
DYCLIENT  DW  200
DATA  ENDS
;сегмент кода
CODE SEGMENT WORD  'CODE'
_BEGIN:
;Начальный код
      CALL  INITTASK                ;инициализировать задачу
      OR AX,AX                      ;CX - границы стека
      JZ _ERR                       ;
      MOV HPREV,SI                  ;номер предыдущего прил.
      MOV HINST,DI                  ;номер для новой задачи
      MOV WORD PTR LPSZCMD,BX       ;ES:BX - адрес командной
строки
      MOV WORD PTR LPSZCMD+2,ES ;
      MOV  CMDSHOW,DX               ;экранный параметр
      PUSH 0                        ;текущая задача
      CALL WAITEVENT                ;очистить очередь событий
      PUSH HINST
      CALL  INITAPP                 ;инициализировать приложения
      OR AX,AX
      JZ _ERR
      CALL MAIN                     ;запуск основной части
_TO_OS:
      MOV AH,4CH
      CALL  DOS3CALL                ; выйти из программы
_ERR:
      CALL  BEEP
      JMP SHORT _TO_OS
;основная процедура
;*****************************************************
MAIN  PROC
; Регистрация класса окна
;стиль окна NULL - стандартное окно
      MOV      WNDCLASS.STYLE,0
;процедура обработки
      LEA      BX,WNDPROC
      MOV      WORD PTR WNDCLASS.LPFNWNDPROC, BX
      MOV      BX,CS
      MOV      WORD PTR WNDCLASS.LPFNWNDPROC+2, BX
;-------------------------------
;резервные байты в конце резервируемой структуры
      MOV      WNDCLASS.CBCLSEXTRA, 0
;резервные байты в конце структуры для каждого окна
      MOV      WNDCLASS.CBWNDEXTRA, 0
```

```
;иконка окна отсутствует
      MOV      WNDCLASS.HICON, O
;номер запускаемой задачи
      MOV      AX,HINST
      MOV      WNDCLASS.HINSTANCE,AX
;определить номер стандартного курсора
      PUSH     O
      PUSH     DS
      PUSH     CURSOR
      CALL     LOADCURSOR
      MOV      WNDCLASS.HCURSOR, AX
;определить номер стандартного объекта
      PUSH     0            ;WHITE_BRUSH
      CALL     GETSTOCKOBJECT
;цвет фона
      MOV      WNDCLASS.HBRBACKGROUND, AX
;имя меню из файла ресурсов (отсутствует = NULL)
      MOV      WORD PTR WNDCLASS.LPSZMENUNAME, O
      MOV      WORD PTR WNDCLASS.LPSZMENUNAME+2, 0
;указатель на строку, содержащую имя класса
      LEA      BX,CLAS_NAME
      MOV      WORD PTR WNDCLASS.LPSZCLASSNAME,BX
      MOV      WORD PTR WNDCLASS.LPSZCLASSNAME+2, DS
;вызов процедуры регистрации
      PUSH DS    ;указатель на
      LEA DI,WNDCLASS
      PUSH DI    ;структуры WNDCLASS
      CALL REGISTERCLASS
      CMP AX,0
      JNZ _OK1
;звуковой сигнал при ошибке
      CALL BEEP
      RET            ;ошибка при регистрации
_OK1:
;Создание окна
;адрес строки-имени класса окна
      PUSH DS
      LEA BX,CLAS_NAME
      PUSH BX
;адрес строки-заголовка окна
      PUSH DS
      LEA BX,APP_NAME
      PUSH BX
;стиль окна
      MOV BX,HIGHWORD STYLE
      PUSH BX
```

```
        MOV   BX,LOWWORD STYLE
        PUSH BX
;координата X левого верхнего угла
        PUSH XSTART
;координата Y левого верхнего угла
        PUSH YSTART
;ширина окна
        PUSH DXCLIENT
;высота окна
        PUSH DYCLIENT
;номер окна-родителя
        PUSH 0
;номер (идентификатор) меню окна
        PUSH 0
;номер задачи
        PUSH HINST
;адрес блока параметров окна (нет)
        PUSH 0
        PUSH 0
        CALL  CREATEWINDOW
        CMP AX,0
        JNZ NO_NULL
        CALL BEEP
        RET          ;ошибка при создании окна
;установка для окна состояния видимости (окно или
;пиктограмма) согласно параметру CMDSHOW и его отображение
NO_NULL:
        MOV SI,AX
        PUSH SI
        PUSH CMDSHOW
        CALL SHOWWINDOW
;посылка команды обновления области окна (команда WM_PAINT)
;сообщение посылается непосредственно окну
        PUSH SI
        CALL UPDATEWINDOW
; Цикл ожидания
        LOOP1:
;извлечение сообщения из очереди
        PUSH DS
        LEA BX,MSG . ;указатель на структуру
        PUSH BX      ; сообщения
        PUSH 0
        PUSH 0
        PUSH 0
        CALL  GETMESSAGE
```

```
;проверка - не получено сообщение "выход"
      CMP AX,0
      JZ  NO_LOOP1
;перевод всех пришедших сообщений к стандарту ANSI
      PUSH DS
      LEA BX,MSG
      PUSH BX
      CALL TRANSLATEMESSAGE
;указать WINDOWS передать данное сообщение соответствующему
;окну
      PUSH DS
      LEA BX,MSG
      PUSH BX
      CALL   DISPATCHMESSAGE
;замкнуть цикл (петлю)
      JMP SHORT LOOP1
NO_LOOP1:
      RET
MAIN ENDP
;процедура для заданного класса окон
;WINDOWS передает в эту процедуру параметры:
;HWND - дескриптор (номер) окна, тип WORD
;MES - номер сообщения, тип WORD
;WPARAM - дополнительная информация о сообщении, тип WORD
;LPARAM - дополнительная информация о сообщении, тип DWORD
WNDPROC PROC
      PUSH BP
      MOV  BP,SP
      MOV AX,[BP+0CH]    ;MES - номер сообщения
      CMP AX,2   ;не сообщение ли о закрытии (2 - сообщение о
закрытии)
      JNZ NEXT1
;передать сообщение о закрытии приложения, это сообщение
;будет принято в цикле ожидания, и т.о. приложение завершит
; свой путь
     PUSH O
      CALL POSTQUITMESSAGE
      JMP _QUIT
NEXT1:
      CMP AX, 1      ;WM_CREATE
      JNZ NEXT2
;создание контрольных элементов
;адрес строки-имени класса окна
      PUSH DS
      LEA BX,PRED_CLAS
      PUSH BX
```

```
;адрес строки-надписи на кнопке
      PUSH  DS
      LEA  BX,BUT_NAME
      PUSH BX
;стиль окна
      MOV  BX,HIGHWORD  STYLE_BUT
      PUSH BX
      MOV   BX,LOWWORD  STYLE_BUT
      PUSH BX
;координата X левого  верхнего  угла
      PUSH  20
;координата У левого  верхнего  угла
      PUSH  20
;ширина окна
      PUSH  60
;высота окна
      PUSH  20
;номер  окна-родителя
      PUSH   [BP+OEH]
;номер  (идентификатор)  кнопки
      PUSH 1
;номер задачи
      PUSH HINST
;адрес блока  параметров  окна  (нет)
      PUSH O
      PUSH O
      CALL   CREATEWINDOW
NEXT2:
      CMP AX,111H     ;WM_COMMAND
      JNZ NEXT
      CMP WORD PTR [BP+0AH],1  ;идентификатор кнопки
      JNZ NEXT
;кнопка нажата  и  Выход
      PUSH O
      CALL  POSTQUITMESSAGE
NEXT:
;передать сообщение  дальше  WINDOWS
;своего рода правило вежливости - то,  что не обработано
;процедурой обработки предоставляется  для  обработки
;WINDOWS
      PUSH  [BP+OEH]        ;HWND
      PUSH  [BP+OCH]        ;MES - номер сообщения
      PUSH  [BP+0AH]        ;WPARAM
      PUSH  [BP+8]          ;HIGHWORD LPARAM
```

```
        PUSH  [BP+6]           ;LOWWORD LPARAM
        CALL  DEFWINDOWPROC
;*******************************************************
_QUIT:
        POP BP
;вызов процедуры окна всегда дальний, поэтому RETF
        RETF 10                ;освобождаем стек от параметров
WNDPROC ENDP
;звуковой сигнал
BEEP  PROC
        MOV AH,2
        MOV DL,7
        CALL  DOS3CALL
        RET
BEEP ENDP
CODE  ENDS
        END   _BEGIN
```

Рис. 24.4. Пример простейшего органа управления.

Данная программа получена из программы на Рис. 24.3, во-первых, изменением стиля окон, во-вторых, добавлением в окно элемента управления - кнопки. Чтобы упростить анализ, замечу следующее:

1. элемент управления открывается как новое окно, но с предопределенным классом, имена всех таких классов известны и определены в самой операционной системе;
2. при создании окна в ее процедуру управления посылается сообщение WM_CREATE, по приходу которого мы и создаем нашу кнопку;
3. создавая элемент управления, мы указываем системе идентификатор окна, которому она принадлежит;
4. одновременно мы присваиваем нашей кнопке свой идентификатор (1);
5. по приходу в процедуру окна сообщения WM_COMMAND мы проверяем WPARAM, при нажатии кнопки он равен идентификатору кнопки, т.е. в нашем случае 1.

А теперь маленькая хитрость, которую, возможно, Вы уже разгадали. Если Вы пишите программу с помощью какого-нибудь DOS-овского редактора, у Вас будут проблемы при выводе русских букв. Дело в кодировке. Проблема решается весьма просто: зайдите в какой-нибудь редактор текстовых файлов, работающий под WINDOWS, например, редактор Си++, и напечатайте там все нужные русские строки.

VI. Программа-таймер.

Программа, представленная ниже, выводит окно, на котором высвечивается текущее время: чч:мм:сс. Я думаю, что Вы уже способны оценить средства программирование в среде WINDOS. Согласитесь, что под MS DOS подобную программу (а она должна быть резидентной), написать несколько сложнее. Впрочем, я беру на себя смелость заявить, что и значительного облегчения в программировании Вы тоже не найдете.

```
 .286
 .DOSSEG
DGROUP GROUP
DATA, STA
      ASSUME CS:CODE, DS:DGROUP
;внешние процедуры
EXTRN CREATEFONTINDIRECT:FAR
EXTRN SELECTOBJECT:FAR
EXTRN DELETEOBJECT:FAR
EXTRN INVALIDATERECT:FAR
EXTRN BEGINPAINT:FAR
EXTRN ENDPAINT:FAR
EXTRN TEXTOUT:FAR
EXTRN SETTIMER:FAR
EXTRN INITTASK:FAR
EXTRN INITAPP:FAR
EXTRN WAITEVENT:FAR
EXTRN DOS3CALL:FAR
EXTRN REGISTERCLASS:FAR
EXTRN LOADCURSOR:FAR
EXTRN GETSTOCKOBJECT:FAR
EXTRN GETMESSAGE:FAR
EXTRN TRANSLATEMESSAGE:FAR
EXTRN DISPATCHMESSAGE:FAR
EXTRN CREATEWINDOWEX:FAR
EXTRN UPDATEWINDOW:FAR
EXTRN SHOWWINDOW:FAR
EXTRN POSTQUITMESSAGE:FAR
EXTRN DEFWINDOWPROC:FAR
;шаблоны
;шаблон для установки фонтов
LOGFON                 STRUCT
_HEIGHT                   DW     0
_WIDTH                    DW     0
_ESCAPEMENT               DW     0
_ORIENTATION              DW     0
_WEIGHT                   DW     0
_ITALIC                   DB     0
_UNDERLINE                DB     0
_STRIKEOUT                DB     0
_CHARSET                  DB     0
_OUTPRECISION             DB     0
_CLIPPRECISION            DB     0
_QUALITY                  DB     0
```

```
_PITCHANDFAMILY         DB     0
_FACENAME               DB     32 DUP(0)
LOGFON              ENDS
;структура для класса окна
WNDCL               STRUCT
STYLE            DW 0  ;стиль класса окна
LPFNWNDPROC      DD 0  ;указатель на процедуру обработки
CBCLSEXTRA       DW 0
CBWNDEXTRA       DW 0
HINSTANCE        DW 0
HICON            DW 0
HCURSOR          DW 0
HBRBACKGROUND    DW 0
LPSZMENUNAME     DD 0  ;указатель на строку
LPSZCLASSNAME    DD 0  ;указатель на строку
WNDCL       ENDS
;структура, определяющая прямоугольную область
RECT    STRUCT
LEFT             DW  0
TOP              DW  0
RIGHT            DW  0
BOTTOM           DW  0
RECT   ENDS
;структура для перерисовки прямоугольной области
PAINTSTRUCT            STRUCT
HDC              DW   7
FERASE           DW   ?
RCPAINT          RECT          0
FRESTORE         DW   ?
FINCUPDATE       DW   ?
RGBRESERVED      DB 16 DUP (?)
PAINTSTRUCT     ENDS
;структура•для системных сообщений
MESSA           STRUCT
HWND             DW  7
MESSAGE          DW  7
WPARAM           DW  7
LPARAM           DD  7
TIME             DW  7
X                DW  ?
Y                DW  ?
MESSA           ENDS
;сегмент стека
STA SEGMENT STACK 'STACK'
      DW 2000 DUP(?)
```

```
STA ENDS
;сегмент данных
DATA SEGMENT WORD 'DATA'
      DWORD  0
      WORD   5
      WORD   5 DUP (0)
HPREV   DW ?
HINST   DW ?
LPSZCMD DD ?
CMDSHOW DW ?
HWND DW ?
HDC  DW ?
HFONT DW ?
;буфер для вывода информации
BUFER DB '   :  :  ',0
;структура для задания шрифта
LOGFONT LOGFON O '
;структура для создания класса
WNDCLASS WNDCL O
;структура сообщения
MSG MESSA O
;прямоугольник для вывода текста
RECTA RECT O
;структура для BEGINPAINT
PAINT PAINTSTRUCT O
;имя класса основного окна
CLAS_NAME DB 'PRIVET',O
;заголовок окна
APP_NAME DB 0  ;нет
;тип курсора
CURSOR EQU 00007FOOH
;стиль окна
STYLE EQU 000080000H
STYLE_EX  EQU 000000008H
; параметры окна
XSTART DW  100
YSTART DW  100
DXCLIENT DW 210
DYCLIENT DW 90
;-------------------
DATA ENDS
;сегмент кода
CODE SEGMENT WORD  'CODE'
_BEGIN:
```

```
/Начальный код
      CALL INITTASK                      ;инициализировать задачу
      OR AX,AX                           ;CX - границы стека
      JZ _ERR                            t
      MOV HPREV,SI                       ;номер предыдущего прил.
      MOV HINST,DI                       ;номер для новой задачи
      MOV WORD PTR LPSZCMD,BX            ;ES:BX - адрес командной
строки
      MOV WORD PTR LPSZCMD+2,ES ;
      MOV CMDSHOW,DX                     ;экранный параметр
      PUSH 0                             ;текущая задача
      CALL WAITEVENT                     ;очистить очередь событий
      PUSH HINST
      CALL INITAPP                       ;инициализировать приложения
      OR AX,AX
      JZ _ERR
      CALL MAIN                          ;запуск основной части
_TO_OS:
      MOV AH,4CH
      CALL DOS3CALL                      ; выйти из программы
_ERR:
      CALL BEEP
      JMP SHORT _TO_OS
;основная процедура
;*****************************************************
MAIN PROC
;Регистрация класса окна
/стиль окна NULL - стандартное окно
      MOV      WNDCLASS.STYLE,0
;процедура обработки
      LEA      BX,WNDPROC
      MOV      WORD PTR WNDCLASS.LPFNWNDPROC,BX
      MOV      BX,CS
      MOV      WORD PTR WNDCLASS.LPFNWNDPROC+2, BX
;---------------------------------
/резервные байты в конце резервируемой структуры
      MOV     WNDCLASS.CBCLSEXTRA, O
/резервные байты в конце структуры для каждого окна
      MOV     WNDCLASS.CBWNDEXTRA, O
/иконка окна отсутствует
      MOV     WNDCLASS.HICON, O
;номер запускаемой задачи
      MOV      AX,HINST
      MOV      WNDCLASS.HINSTANCE,AX
```

```
;определить номер стандартного курсора
        PUSH    0
        PUSH    DS
        PUSH    CURSOR
        CALL    LOADCURSOR
        MOV     WNDCLASS.HCURSOR, AX
;определить номер стандартного объекта
        PUSH    0           ;WHITE_BRUSH
        CALL    GETSTOCKOBJECT
;цвет фона
        MOV     WNDCLASS.HBRBACKGROUND, AX
;имя меню из файла ресурсов (отсутствует = NULL)
        MOV     WORD PTR WNDCLASS.LPSZMENUNAME, 0
        MOV     WORD PTR WNDCLASS.LPSZMENUNAME+2, 0
;указатель на строку, содержащую имя класса
        LEA     BX,CLAS_NAME
        MOV     WORD PTR WNDCLASS.LPSZCLASSNAME, BX
        MOV     WORD PTR WNDCLASS.LPSZCLASSNAME+2,DS
;вызов процедуры регистрации
        PUSH DS     ; указатель на
        LEA DI,WNDCLASS
        PUSH DI     ;структуры WNDCLASS
        CALL REGISTERCLASS
        CMP AX,0
        JNZ _OK1
;звуковой сигнал при ошибке
        CALL BEEP
        RET             ;ошибка при регистрации
_OK1:
;Создание окна
;расширенный стиль - всегда поверх других
        MOV   BX,HIGHWORD STYLE_EX
        PUSH BX
        MOV   BX,LOWWORD STYLE_EX
        PUSH BX
;адрес строки-имени класса окна
        PUSH DS
        LEA BX,CLAS_NAME
        PUSH BX
;адрес строки-заголовка окна
        PUSH DS
        LEA BX,APP_NAME
        PUSH BX
```

```
;стиль окна
      MOV BX,HIGHWORD STYLE
      PUSH BX
      MOV  BX,LOWWORD STYLE
      PUSH BX
;координата X левого верхнего угла
      PUSH XSTART
;координата Y левого верхнего угла
      PUSH YSTART
;ширина окна
      PUSH DXCLIENT
; высота окна
      PUSH DYCLIENT
;номер окна-родителя
      PUSH O
;номер (идентификатор) меню окна
      PUSH O
;номер задачи
      PUSH HINST
;адрес блока параметров окна (нет)
      PUSH 0  •
      PUSH O
      CALL  CREATEWINDOWEX
      CMP AX,0
      JNZ NO_NULL
      CALL BEEP
      RET         ;ошибка при создании окна
;установка для окна состояния видимости (окно или
;пиктограмма) согласно параметру CMDSHOW и его отображение
NO_NULL:
      MOV HWND,AX
      MOV SI,AX
      PUSH SI
      PUSH CMDSHOW
      CALL SHOWWINDOW
; посылка команды обновления области окна (команда WM_PAINT)
;сообщение посылается непосредственно окну
      PUSH SI
      CALL UPDATEWINDOW
;заполнение структуры фонта (не нулевые поля)
      MOV     LOGFONT._HEIGHT, 60
      MOV     LOGFONT._WEIGHT,400              ;FW_NORMAL
      MOV     LOGFONT._PITCHANDFAMILY,20H      ; FF_SWISS
```

```
;Цикл ожидания
      LOOP1:
;извлечение сообщения из очереди
      PUSH DS
      LEA BX,MSG  ;указатель на структуру
      PUSH BX     ;сообщения
      PUSH O
      PUSH O
      PUSH O
      CALL GETMES.SAGE
;проверка - не получено сообщение "выход"
      CMP AX,O
      JZ  NO_LOOP1
;перевод всех пришедших сообщений к стандарту ANSI
      PUSH DS
      LEA BX,MSG
      PUSH BX
      CALL TRANSLATEMESSAGE
;указать WINDOWS передать данное сообщение соответствующему
;окну
      PUSH DS
      LEA BX,MSG
      PUSH BX
      CALL  DISPATCHMESSAGE
;замкнуть цикл (петлю)
      JMP SHORT LOOP1
NO_LOOP1:
      RET
MAIN ENDP
;процедура для заданного класса окон
;WINDOWS передает в эту процедуру параметры:
;HWND - дескриптор (номер) окна, тип WORD
;MES - номер сообщения, тип WORD
;WPARAM - дополнительная информация о сообщении, тип WORD
;LPARAM - дополнительная информация о сообщении, тип DWORD
WNDPROC PROC
      PUSH BP
      MOV  BP,SP
      MOV AX,[BP+0CH]    /MES - номер сообщения
      CMP AX,2  /не сообщение ли о закрытии (2-сообщение о
закрытии)
      JNZ NEXT1
/передать сообщение о закрытии приложения, это сообщение будет
/принято в цикле ожидания, и т.о. приложение завершит свой путь
      PUSH O
      CALL POSTQUITMESSAGE
      JMP _QUIT
```

```
NEXT1:
      CMP AX,1           ;WM_CREATE - создание окна
      JNZ NEXT2
;установим таймер
      PUSH  [BP+0EH]
      PUSH 1
      PUSH 1000          ;вызывать через 1000 миллисекунды
      PUSH 0
      PUSH 0
      CALL  SETTIMER
      JMP   _QUIT
NEXT2:
      CMP AX,111H       ;WM_COMMAND - сообщение-команда
      JNZ NEXT3
      JMP _QUIT
NEXT3:
      CMP AX,113H       ;WM_TIMER - сообщение таймера
      JNZ NEXT4
;послать команду перерисовки области всего окна
;это приведет к посылке в очередь сообщения WM_PAINT
      PUSH  [BP+0EH]
      PUSH 0
      PUSH 0
      PUSH 1
      CALL  INVALIDATERECT
      JMP _QUIT
NEXT4:
      CMP AX,0FH        ;WM_PAINT - перерисовка окна
      JNZ NEXT5
      PUSH  [BP+0EH]
      PUSH DS
      LEA BX,PAINT
      PUSH BX
      CALL BEGINPAINT
      MOV HDC,AX
;———установка шрифтов
      PUSH DS
      LEA   BX,LOGFONT
      PUSH BX
      CALL  CREATEFONTINDIRECT
;--
      MOV HFONT,AX
      PUSH HDC
      PUSH AX
      CALL  SELECTOBJECT
```

```
;--читать часы реального времени
      MOV AH,2
      INT  1AH
      CALL _BCD
;-- вывести строку
      PUSH HDC
      PUSH 1
      PUSH 1
      PUSH DS
      LEA  BX,BUFER
      PUSH BX
      PUSH 8     ;количество символов
      CALL TEXTOUT
;--
      PUSH HDC
      PUSH HFONT
      CALL  SELECTOBJECT
;--
      PUSH AX
      CALL  DELETEOBJECT
;__
      PUSH HDC
      PUSH DS
      LEA  BX,PAINT
      PUSH BX
      CALL ENDPAINT
      JMP _QUIT
NEXT5:
NEXT:
;передать сообщение дальше WINDOWS
;своего рода правило вежливости - то, что не обработано
;процедурой обработки предоставляется для обработки
/WINDOWS
      PUSH  [BP+OEH]        ;HWND
      PUSH  [BP+OCH]        ;MES - номер сообщения
      PUSH  [BP+OAH]        ;WPARAM
      PUSH  [BP+8]          ;HIGHWORD LPARAM
      PUSH  [BP+6]          ;LOWWORD LPARAM
      CALL  DEFWINDOWPROC
;*****************************************************
_QUIT:
      POP  BP
/вызов процедуры окна всегда дальний, поэтому RETF
      RETF  10              ;освобождаем стек от параметров
WNDPROC  ENDP
```

```
;звуковой сигнал
BEEP PROC
      MOV AH,2
      MOV DL,7
      CALL DOS3CALL
      RET
BEEP ENDP
;преобразование числа в BCD-формате в шаблон для печати
;времени по часам реального времени
_BCD PROC
;часы
      MOV AL,CH
      SHR AL,4
      ADD AL,48
      AND CH,00001111B
      CMP CH,10
      JB  _OK1
      ADD AL,1
      SUB CH,10
_OK1:
      ADD CH,48
      MOV BYTE PTR BUFER,AL
      MOV BYTE PTR BUFER+1,CH
;минуты
      MOV AL,CL
      SHR AL,4
      ADD AL,48
      AND CL,00001111B
      CMP CL,10
      JB  _OK2
      ADD AL,1
      SUB CL,10
_OK2:
      ADD CL,48
      MOV BYTE PTR BUFER+3,AL
      MOV BYTE PTR BUFER+4,CL
;секунды
      MOV AL,DH
      SHR AL,4
      ADD AL,48
      AND DH,00001111B
      CMP DH,10
      JB  _OK3
      ADD AL,1
      SUB DH,10
```

```
_OK3:
       ADD  DH,48
       MOV  BYTE  PTR  BUFER+6,AL
       MOV  BYTE  PTR  BUFER+7,DH
       RETN
_BCD  ENDP
CODE  ENDS
       END   _BEGIN
```

Рис. 24.5. Программа-таймер.

Программа достаточно прокомментирована. Описание же функций, которые появились в данной программе, Вы найдете и сами. Обращаю внимание только на то, что структура программы осталась неизменной по сравнению с предыщущими. Чтобы облегчить анализ, предлагаю учесть следующее:

1. При создании (WM_CREATE) окна мы устанавливаем и таймер. Сообщение таймера (WM_TIMER) будет приниматься опять же процедурой окна.
2. Перед выводом строки в окно мы устанавливаем текущий шрифт. После вывода - удаляем созданный шрифт. Структура для шрифта (LOGFONT) заполняется нами заранее.
3. По приходу сообщения WM_TIMER мы даем команду для перерисовки окна. После чего в очередь сообщений автоматически ставится сообщение WM_PAINT.

Глава 25. 32-битное программирование в WINDOWS. (Программируем в WINDOWS 95-98.)

- Тем, кто хорошо знаком с пятым измерением, ничего не стоит раздвинуть помещение до желательных пределов. Сказку вам более, уважаемая госпожа, до черт знает каких пределов!

М.А. Булгаков.
Мастер и Маргарита.

До сих пор мы писали программы фактически для Windows 3.1. И хотя эти программы прекрасно будут работать и в Windows 95, все же они «не родные» для операционной системы. Сейчас наша задача - научиться писать программы для Windows 95, т.е. программы, работающие в 32-битном режиме.

В ОС Windows 95 реализована так называемая плоская, или линейная, модель памяти. Вся память рассматривается как один сегмент. Адрес любой ячейки является смещением в этом сегменте. Кроме того, в Windows 95 используется страничная адресация (см. главу 20), позволяющая эффективно контролировать разделение задач. По этой причине можно забыть о моделях памяти, объеме данных и стека. Программисты на Си должны оценить это по достоинству.

К сожалению, на момент написания этой главы, у меня отсутствовали библиотеки для работы в Windows 95 для ассемблера фирмы Microsoft[66], поэтому я буду пользоваться библиотеками для Турбо-ассемблера (5.0) и Си++ (вер.5.0). Описание новых функций API можно найти в руководствах для Си++.

Обращаю Ваше внимание, что во всех дальнейших программах порядок посылки параметров обратный (справа налево).

I. Консольный режим.

Для поддержки задач, работающих в текстовом режиме, в Windows 95 появился новый режим, называемый консольным. Консоль — это текстовое окно, такое же, как для программ MS DOS. Однако программы, использующие консольный режим, являются полноценными 32-битными программами, которые, как и другие программы, могут пользоваться функциями API. Консольный режим очень прост и позволяет быстро писать простые прикладные программы для работы в среде Windows 95. Для работы в консольном режиме было добавлено несколько API-функций. Известная, наверное, уже всем, программа Far написана как раз в консольном режиме.

---

[66] Необходима версия MASM 6.14.

Обращаю внимание на то, что в данном разделе в вызовах функций API (но не обычные функции Си) следует различать прописные и заглавные буквы. Объясняется это просто: функции API будут вызываться во время выполнения программы из уже готовых динамических библиотек и должны иметь точно такое имя, как в этих библиотеках. По этой же причине при трансляции мы используем ключ /ml.

Ниже (Рис. 24.6) представлена элементарная программа, открывающая окно консоли и ожидающая нажатие клавиши. Программа использует 32-битные библиотеки Си++(5.0): c0x32.obj, cw32.1ib, import32.1ib. Для трансляции можно по-прежнему использовать MASM 6.0, но для компоновки требуется программа tlink32.exe. Эту программу можно найти в пакете Си++. Если наша программа называется consol1.asm, то для ее трансляции в consol1 .exe требуются команды:

```
masm /Ml consoll
tlink32 c0x32+consoll,consoll,,cw32 imort32
```

Библиотеки cOx32.obj, import32.lib и cw32.1ib должны находиться в текущем каталоге. Подобная программа, использующая Си библиотеки для MS DOS, была нами рассмотрена в главе 15.

```
.386P
;плоская модель памяти
.MODEL FLAT
EXTRN    AllocConsole:NEAR
EXTRN    FreeConsole:NEAR
EXTRN    _GETCH:NEAR
PUBLIC   _MAIN
_TEXT    SEGMENT DWORD PUBLIC USE32  'CODE'
;процедура, вызываемая из заголовка программы cOx32.obj
_MAIN    PROC     NEAR
;освободить память и инициализировать консоль
      CALL       FreeConsole
      CALL       AllocConsole
;ждем нажатие клавиши
      CALL       _GETCH
;освободить память
      CALL       FreeConsole
;возвратить управление в заголовок программы
      RET
_MAIN  ENDP
_TEXT  ENDS
      END
```

Рис. 24.6. Элементарная консольная программа.

Кроме используемых в программе функций, имеются еще следующие функции API для работы с консолью (в Си нотации):

Текст заголовка для окна консоли задается следующей функцией:

int SetConsoleTitle(str), str - указатель на строку, которая станет заголовком.

Для получения стандартного дескриптора консоли используется функция:

int GetStdHandle(param), param - принимает значение:

STD_INPUT_HANDLE - -10 - для ввода

STD_OUTPUT_HANDLE - -11 - для вывода

STD_ERROR_HANDLE - -12 - для сообщения об ошибках.

Вывод текста на консоль:

int WriteConsole(handl, str, len, len1, noused), handl - дескриптор вывода, str - указатель на строку, len количество символов, которое необходимо вывести, len1 - указатель на двойное слово, куда помещается количество реально выведенных символов, noused - неиспользуемый параметр.

Ввод с консоли:

int ReadConsole(handl, buf, len, len1, noused), handl -дескриптор консоли, buf - указатель на буфер приема, len - количество вводимых символов, len1 - указатель на двойное слово, которое будет содержать количество реально введенных символов, noused - неиспользуемый параметр.

Установка позиции курсора:

int SetConsoleCursorPosition(handl, coord), handl -дескриптор консоли, coord - указатель на структуру -

x db ?

y db ?

Установка цветов текста и фона:

int SetConsoleTextAttribute(handl, color),

handl - дескриптор консоли, color - слово, определяющее цвет фона и текста, получается путем операции OR над следующими параметрами

FOREGROUND_BLUE 0x0001 - синий текст,

FOREGROUND_GREEN 0x0002 - зеленый текст,

FOREGROUND_RED 0x0004 - красный текст,

FOREGROUND_INTENSITY 0x0008 - повышенная яркость текста,

BACKGROUND_BLUE 0x0010 - синий фон,

BACKGROUND_GREEN 0x0020 - зеленый фон,

BACKGROUND_RED 0x0040 - красный фон,

BACKGROUND_INTENSITY 0x0080 - повышенная яркость фона.

Обработка мыши и клавиатуры:

int ReadConsoleInput(handl, buf, num, num1),

handl - дескриптор консоли, buf - указатель на структуру (см. ниже), num - количество информационных записей о событиях, num1 - количество реально возвращенных записей. Функция ReadMouseInput() читает одну или несколько записей о входных событиях в буфер. Структура буфера следующая (в Си нотации):

```
typedef struct INPUT_RECORD
{
     WORD EventType;
     union
     {
     KEY_EVENT_RECORD KeyEvent;      && нажатие клавиши
     MOUSE_EVENT_RECORD MouseEvent; && событие мыши
     WINDOW_BUFFER_SIZE_RECORD WindowBufferSizeEvent; &&изменение размера
     MENU_EVENT_RECORD MenuEvent;    && используется системой
     FOCUS_EVENT_RECORD FocusEvent; && используется системой
     } Event;
} INPUT_RECORD;
```

Рассмотрим пример следующей консольной программы. Содержимое текстового файла выводится на консоль. Программа подробно прокомментирована. Необходимо сделать, однако, несколько замечаний:

1. В программе используются как функции API, так и обычные Си функции (открыть файл, прочитать в буфер, закрыть файл).

2. Вызов обычных функций осуществляется согласно нотации Си, т.е. с очисткой стека после вызова (см. главу 15).

3. Порядок помещения параметров в стек отличен от того, которым мы до сих пор пользовались.

4. Обращаю Ваше внимание на следующий интересный факт. Для данных мы выделили отдельный сегмент. Однако мы знаем, что структура памяти все равно состоит из одного глобального сегмента. В принципе можно было бы использовать только один сегмент _TEXT. Однако здесь есть одно "но". Попытка осуществить запись в сегмент кода вызовет во время выполнения программы ошибку. Дело в том, что защита памяти в Windows 95 осуществляется посредством страничной адресации (см. главу 20). И те данные, которые мы поместим в сегмент кода, будут открыты только для чтения. Если данные предполагается менять, то для них следует отвести сегмент данных. В этом случае они будут открыты как для чтения, так и для записи.

```
.386P
;плоская модель
.MODEL FLAT
;внешние процедуры
EXTRN    GetStdHandle:near
EXTRN    AllocConsole:near
EXTRN    FreeConsole:near
EXTRN    WriteConsoleA:near
EXTRN    _CLOSE:near
EXTRN    _OPEN:near
EXTRN    _READ:near
EXTRN    _GETCH:near
PUBLIC   _MAIN
```

```
;сегмент кода
_TEXT   SEGMENT DWORD PUBLIC USE32 'CODE'
;процедура, с которой выполняется код программы
_MAIN   PROC    NEAR
;освободить память и инициализировать консоль
        CALL        FreeConsole
        CALL        AllocConsole
;получить дескриптор вывода на консоль
        PUSH        -11
        CALL        GetStdHandle
        MOV         EDI,EAX
;открыть файл для чтения
        PUSH        8002H ;O_BINARY OR O_RDVR
        PUSH        OFFSET TEX
        CALL        _OPEN
        ADD         ESP,8     ;восстановить стек
        MOV         ESI,EAX   /сохранить дескриптор файла
        CMP         EAX,-1    ;не ошибка ли
        JNE          SHORT LOO
;выйти и указать код ошибки
        MOV         EAX,1
        JMP •       SHORT _END
LOO:
;читаем из файла в буфер
        PUSH -      100           ;читаем сто байт
        PUSH        OFFSET MSG ;адрес буфера (32-байтный)
        PUSH        ESI           ;дескриптор
        CALL        _READ
        ADD         ESP,12        ;восстановить стек
        MOV         EBX,EAX       ;сколько прочли байт
;выведем буфер на консоль
        PUSH        0             ;пустой параметр
        PUSH        OFFSET NUM ;здесь количество выведенных байт
        PUSH        EBX           /количество выводимых байт
        PUSH        OFFSET MSG ;адрес строки (буфера)
        PUSH        EDI           /дескриптор консоли
        CALL        WriteConsoleA
        CMP         EBX, 100   ;если меньше, то файл закончился
        JE          SHORT LOO
; закрыть файл
        PUSH        ESI
        CALL        _CLOSE
        POP         ECX
```

```
;ждем нажатие  клавиши
      CALL        _GETCH
;освободить память
      CALL        FreeConsole
;выход с кодом О
      XOR         EAX,EAX
_END:
      RET
_MAIN   ENDP
_TEXT   ENDS
;сегмент данных
_DATA   SEGMENT  DWORD PUBLIC USE32  'DATA'
TEX     DB       "TEST.TXT",0
NUM     DD       ?
MSG     DB       100 DUP(O)
_DATA   ENDS
      END
```

Рис. 24.7. Вывод текстового файла в консоль.

В заключение разбора программы на Рис. 24.7 отмечу, что имя текстового файла может быть и "длинным". Используемая для открытия файла функция _open хоть и пришла из старого Си, но работает в соответствии с требованием времени. В имени файла, разумеется, могут содержаться и русские буквы. В последнем случае не забудьте только о возможном искажении кодировки (см. выше).

II. 32-х битное программирование.

При рассмотрении консольного режима мы использовали три разнородные компоненты: MASM.EXE (Microsoft), tlink32.exe - 32-битный редактор связи (Borland) и библиотеки из пакета Ci++ Borland 5.O. Не имея новых версий MASM, в данном разделе мы вынуждены перейти к турбо ассемблеру. Я использую TASM версии 5.0, tlink32.exe, уже нами знакомую, и библиотеку import32.lib из пакета TASM. При этом мы по-прежнему придерживаемся старого стиля: не используем include-файлы, не используем сокращенное описание сегментов.

Прежде всего заметим, что консольный режим, изложенный в предыдущем разделе, может использоваться и в программах, описанных ниже. Появление в них команд консольного режима никак не скажется на алгоритме их компилирования.

Ниже представлена простая программа, написанная так, как должно писать для Windows 95. Назовем программу prog8. Трансляция осуществляется выполнением двух строк:

```
TASM32 /ml prog8
tlink32 prog8,prog8,prog8, import32
```

В первую очередь обращаю Ваше внимание на то, что бросается в глаза. Все внешние процедуры записаны с использованием и прописных и заглавных букв. Объяс-

нение этому очень простое: связывание процедур осуществляется динамически, т.е. во время выполнения программы. В динамической же библиотеке они записаны именно так, как у нас в программе. Сказанное в равной мере относится и к предыдущей программе (Рис. 24.7).

```
.386P
;плоская модель
;в стек с права на лево
.MODEL FLAT

;внешние процедуры
EXTRN           Beep:NEAR
EXTRN           CreateWindowExA:NEAR
EXTRN           DefWindowProcA:NEAR
EXTRN           DispatchMessageA:NEAR
EXTRN           ExitProcess:NEAR
EXTRN           GetMessageA:NEAR
EXTRN           GetModuleHandleA:NEAR
EXTRN           LoadCursorA:NEAR
EXTRN           LoadIconA:NEAR
EXTRN           PostQuitMessage:NEAR
EXTRN           RegisterClassA:NEAR
EXTRN           ShowWindow:NEAR
EXTRN           TranslateMessage:NEAR
EXTRN           UpdateWindow:NEAR
PUBLIC  WNDPROC

;структуры
MSGSTRUCT  STRUC
MSHWND          DD    ?
MSMESSAGE       DD    ?
MSWPARAM        DD    ?
MSLPARAM        DD    ?
MSTIME          DD    ?
MSPT            DD    ?
MSGSTRUCT  ENDS
;---------
WNDCLASS  STRUC
CLSSTYLE            DD   ?
CLSLPFNWNDPROC      DD   ?
CLSCBCLSEXTRA       DD   ?
CLSCBWNDEXTRA       DD   ?
CLSHINSTANCE        DD   ?
CLSHICON            DD   ?
```

```
CLSHCURSOR          DD  ?
CLSHBRBACKGROUND    DD  ?
CLSLPSZMENUNAME     DD  ?
CLSLPSZCLASSNAME    DD  ?
WNDCLASS ENDS
;сегмент данных
_DATA SEGMENT DWORD PUBLIC USE32  'DATA'
NEWHWND            DD 0
MSG                MSGSTRUCT   <?>
WC                 WNDCLASS    <?>
HINST              DD 0
TITLENAME          DB  'WIN32, SIMPLE ASSEMBLY PROGRAM',0
CLASSNAME          DB  'CLASS32',0
_DATA  ENDS
сегмент кода
_TEXT  SEGMENT DWORD PUBLIC USE32  'CODE'
START:
      PUSH     0
      CALL     GetModuleHandleA
      MOV      [HINST], EAX
REG_CLASS:
;заполнить структуру окна
; стиль CS_HREDRAW + CS_VREDRAW + CS_GLOBALCLASS
      MOV      [WC.CLSSTYLE] ,4003H
;процедура обработки сообщений
      MOV      [WC.CLSLPFNWNDPROC], OFFSET WNDPROC
      MOV      [WC.CLSCBCLSEXTRA], 0
      MOV      [WC.CLSCBWNDEXTRA], 0
      MOV      EAX, [HINST]
      MOV      [WC.CLSHINSTANCE], EAX
;---       -
      PUSH     32512    ;IDI_APPLICATION
      PUSH     0
      CALL     LoadIconA
      MOV      [WC.CLSHICON], EAX
;----------
      PUSH     32512    ;IDC_ARROW
      PUSH     0
      CALL     LoadCursorA
      MOV      [WC.CLSHCURSOR], EAX
;----------
      MOV      [WC.CLSHBRBACKGROUND], 17   ;цвет окна
      MOV      DWORD PTR [WC.CLSLPSZMENUNAME], 0
      MOV      DWORD PTR [WC.CLSLPSZCLASSNAME], OFFSET CLASSNAME
```

```
        PUSH      OFFSET WC
        CALL      RegisterClassA
;создать окно зарегистрированного класса
        PUSH      0
        PUSH      [HINST]
        PUSH      0
        PUSH      0
        PUSH      400                 ; DY
        PUSH      400                 ; DX
        PUSH      100                 ; Y
        PUSH      100                 ; X
        PUSH      000CF0000H          ;WS_OVERLAPPEDWINDOW
        PUSH      OFFSET TITLENAME      ;имя окна
        PUSH      OFFSET CLASSNAME      ;имя класса
        PUSH      0
        CALL      CreateWindowExA
;проверка на ошибку
        CMP       EAX,0
        JZ        _ERR
        MOV       [NEWHWND], EAX
        PUSH      1        ;SW_SHOWNORMAL
        PUSH      [NEWHWND]
        CALL      ShowWindow
        PUSH      [NEWHWND]
        CALL      UpdateWindow
;петля обработки сообщений
MSG_LOOP:
        PUSH      0
        PUSH      0
        PUSH      0
        PUSH      OFFSET MSG
        CALL      GetMessageA
        CMP       AX, 0
        JE        END_LOOP
        PUSH      OFFSET MSG
        CALL      TranslateMessage
        PUSH      OFFSET MSG
        CALL      DispatchMessageA
        JMP       MSG_LOOP
END_LOOP:
        PUSH      [MSG.MSWPARAM]
        CALL      ExitProcess
_ERR:
        CALL    BEE
        JMP       END_LOOP
```

```
;-----------------------------------------------------------------
;процедура окна
;расположение параметров в стеке
;  [BP+014H]       ;LPARAM
;  [BP+10H]        ;WAPARAM
;  [BP+0CH]        ;MES
;  [BP+8]          ;HWND
WNDPROC            PROC
        PUSH   EBP
        MOV    EBP,ESP
        PUSH   EBX
        PUSH   ESI
        PUSH   EDI
        CMP    DWORD PTR [EBP+0CH],2      ;WM_DESTROY
        JE     WMDESTROY
        CMP    DWORD PTR [EBP+0CH],1      ;WM_CREATE
        JE     WMCREATE
        CMP    DWORD PTR [EBP+0CH],201H ;левая кнопка
        JE     LBUTTON
        CMP    DWORD PTR [EBP+OCH] ,204H ;правая кнопка
        JE     RBUTTON
        JMP    DEFWNDPROC
;нажатие правой кнопки приводит к закрытию окна
RBUTTON:
        JMP     WMDESTROY
;нажатие левой кнопки вызывает звуковой сигнал
LBUTTON:
        CALL   BEE
        MOV    EAX,0
        JMP    FINISH
WMCREATE:
        MOV       EAX, 0
        JMP       FINISH
DEFWNDPROC:
        PUSH      DWORD  PTR  [EBP+14H]
        PUSH      DWORD  PTR  [EBP+10H]
        PUSH      DWORD  PTR  [EBP+OCH]
        PUSH      DWORD  PTR  [EBP+08H]
        CALL      DefWindowProcA
        JMP       FINISH
WMDESTROY:
        PUSH       0
        CALL      PostQuitMessage
        MOV       EAX, 0
```

```
FINISH:
      POP   EDI
      POP   ESI
      POP   EBX
      POP   EBP
      RET   16
WNDPROC           ENDP
;------------------------------------------------------------
;процедура короткого звукового сигнала
BEE  PROC
      MOV      ECX,100
      PUSH     ECX
_OP:
      PUSH     0
      PUSH     0
;библиотечная процедура, дающая короткий звуковой сигнал
      CALL     Beep
      POP      ECX
      DEC      ECX
      PUSH     ECX
      LOOPD    _OP
      POP  ECX
      RET
BEE  ENDP
_TEXT  ENDS
      END  START
```

Рис. 24.8. Простая программа для WINDO WS95 с 32-битной адресацией.

Обращаю внимание на одну особенность программы на Рис.24.8 - она начинается с вызова функции GetModuleHandleA. Наши 16-битные программы начинались, как Вы помните, с других функций. Однако стало удобнее, не правда ли?

III. Использование ресурсов.

Вручную создавать кнопки, списки, меню и т.д. - дело довольно утомительное. Но, к счастью, у нас имеется мощный инструмент - ресурсы, которые можно создавать отдельно и затем подключать к EXE-файлам. Перед нами стоят три вопроса:

1. Как создавать ресурсы?
2. Как подключать ресурсы?
3. Как использовать ресурсы в программе?

Начнем отвечать по порядку.

1. Ресурсы хранятся в текстовом файле с расширением RC, который можно заполнять обычным текстовым редактором. Я советовал бы первый ресурс создать именно так. В дальнейшем удобнее, однако, использовать редактор ресурсов, ко-

торый имеется, например, в Borland Си. В нем создание файла ресурсов осуществляется визуальными средствами. Для простоты возьмем только один ресурс меню и только с двумя пунктами. Файл ресурсов в этом случае будет иметь вид:

```
MENUP MENU
{
   POPUP "&Файлы"
  {
         MENUITEM "&Пункт меню", 101
         MENUITEM "E&xit", 102
  }
}
```

Обратите внимание на следующие моменты. Наш ресурс называется MENUP. По этому имени будет осуществлять обращение к данному ресурсу. Каждому пункту меню присваивается свой номер. По этому номеру происходит идентификация выбранного пункта меню. Горячая клавиша отмечается посредством знака &.

2. Перед подключением ресурса его следует откомпилировать с помощью подходящего компилятора ресурсов. Я пользовался компилятором BRC.EXE. Предположим, файл ресурсов называется R1.RC, тогда строка выглядит так: BRC-32 –r R1. В результате появляется откомпилированный файл R1.RES. Для того чтобы файл ресурсов был включен в откомпилированный файл, его следует указать в строке для tlik32. Пусть объектный файл называется progl, тогда: tlink32 progl,progl,progl,imort32,,r1 и в файл prog1.exe будет включен ресурс R1.RES.
3. Для того чтобы меню появилось в окне, при регистрации класса окон следует указать на строку с именем меню (у нас это MENUP). При выборе пункта меню в функцию окна приходит сообщение WM_COMMAND. При этом в младшем слове WPARAM помещается номер-значение выбранного пункта меню.

На Рис. 24.9 представлена программа, полученная из программы на Рис. 24.8 добавлением ресурса - меню. По сравнению с предыщущей программой здесь добавился еще вызов функции MessageBox. Эта функция выдает на экран сообщение.

```
.386P
;плоская модель
.MODEL FLAT
;внешние процедуры
EXTRN          Beep:NEAR
EXTRN          CreateWindowExA:NEAR
EXTRN          DefWindowProcA:NEAR
EXTRN          DispatchMessageA:NEAR
EXTRN          ExitProcess:NEAR
EXTRN          GetMessageA:NEAR
EXTRN          GetModuleHandleA:NEAR
EXTRN          LoadCursorA:NEAR
```

```
EXTRN            LoadIconA:NEAR
EXTRN            PostQuitMessage:NEAR
EXTRN            RegisterClassA:NEAR
EXTRN            ShowWindow:NEAR
EXTRN            TranslateMessage:NEAR
EXTRN            UpdateWindow:NEAR
EXTRN            MessageBoxA:NEAR
PUBLIC WNDPROC
;структуры
MSGSTRUCT  STRUC
MSHWND          DD    7
MSMESSAGE       DD    7
MSWPARAM        DD    7
MSLPARAM        DD    7
MSTIME          DD    7
MSPT            DD    7
MSGSTRUCT  ENDS
;---------
WNDCLASS  STRUC
CLSSTYLE            DD    7
CLSLPFNWNDPROC      DD    7
CLSCBCLSEXTRA       DD    7
CLSCBWNDEXTRA       DD    7
CLSHINSTANCE        DD    7
CLSHICON            DD    7
CLSHCURSOR          DD    7
CLSHBRBACKGROUND    DD    7
CLSLPSZMENUNAME     DD    7
CLSLPSZCLASSNAME    DD    7
WNDCLASS  ENDS
;сегмент данных
_DATA SEGMENT DWORD PUBLIC USE32  'DATA'
NEWHWND          DD 0
MSG              MSGSTRUCT    <?>
WC               WNDCLASS     <?>
HINST            DD 0
TITLENAME        DB  'WIN32, ASSEMBLY PROGRAM WITH MENU',0
CLASSNAME        DB  'CLASS32',0
MENU             DB  'MENUP',0
TEXTM            DB  'Выбран пункт меню',0
TEXTMC           DB  'Сообщение',0
_DATA  ENDS
;сегмент кода
_TEXT   SEGMENT DWORD PUBLIC USE32 'CODE'
```

```
START:
        PUSH      0
        CALL      GetModuleHandleA
        MOV       [HINST], EAX
REG_CLASS:
;заполнить структуру окна
;стиль CS_HREDRAW + CS_VREDRAW + CS_GLOBALCLASS
        MOV       [WC.CLSSTYLE],4003H
;процедура обработки сообщений
        MOV       [WC.CLSLPFNWNDPROC], OFFSET WNDPROC
        MOV       [WC.CLSCBCLSEXTRA], 0
        MOV       [WC.CLSCBWNDEXTRA], 0
        MOV       EAX, [HINST]
        MOV       [WC.CLSHINSTANCE], EAX
;----------
        PUSH      32512    ;IDI_APPLICATION
        PUSH      0
        CALL      LoadIconA
        MOV       [WC.CLSHICON], EAX
;----------
        PUSH      32512    ;IDC_ARROW
        PUSH      0
        CALL      LoadCursorA
        MOV       [WC.CLSHCURSOR], EAX
;----------
        MOV       [WC.CLSHBRBACKGROUND], 17   ;цвет окна
        MOV       DWORD PTR [WC.CLSLPSZMENUNAME],OFFSET MENU
        MOV       DWORD PTR [WC.CLSLPSZCLASSNAME], OFFSET CLASSNAME
        PUSH      OFFSET WC
        CALL      RegisterClassA
;создать зарегистрированного класса
        PUSH      O
        PUSH      [HINST]
        PUSH      0
        PUSH      0
        PUSH      400              ;: DY
        PUSH      400              ;: DX
        PUSH      100              ;: Y
        PUSH      100              ;: X
        PUSH      OOOCFOOOOH       ;WS_OVERLAPPEDWINDOW
        PUSH      OFFSET TITLENAME    ;имя окна
        PUSH      OFFSET CLASSNAME    ;имя класса
        PUSH      0
        CALL      CreateWindowExA
```

```
;проверка на ошибку
        CMP     EAX, 0
        JZ      _ERR
        MOV     [NEWHWND], EAX
        PUSH    1       ;SW_SHOWNORMAL
        PUSH    [NEWHWND]
        CALL    ShowWindow
        PUSH    [NEWHWND]
        CALL    UpdateWindow
;петля обработки сообщений
MSG_LOOP:
        PUSH    0
        PUSH    0
        PUSH    0
        PUSH    OFFSET MSG
        CALL    GetMessageA
        CMP     AX, 0
        JE      END_LOOP
        PUSH    OFFSET MSG
        CALL    TranslateMessage
        PUSH    OFFSET MSG
        CALL    DispatchMessageA
        JMP     MSG_LOOP
END_LOOP:
        PUSH    [MSG.MSWPARAM]
        CALL    ExitProcess
_ERR:
        CALL    BEE
        JMP     END_LOOP
;--------------------------------------------------------------
;процедура окна
;расположение параметров в стеке
;  [BP+014H]        ;LPARAM
;  [BP+10H]         ;WAPARAM
;  [BP+OCH]         ;MES
;  [BP+8]           ;HWND
WNDPROC          PROC
        PUSH    EBP
        MOV     EBP,ESP
        PUSH    EBX
        PUSH    ESI
        PUSH    EDI
        CMP     DWORD PTR [EBP+0CH],2     ;WM_DESTROY
        JE      WMDESTROY
```

```
        CMP    DWORD PTR  [EBP+0CH],1      ;WM_CREATE
        JE     WMCREATE
        CMP    DWORD PTR  [EBP+0CH],201H ;левая кнопка
        JE     LBUTTON
        CMP    DWORD PTR  [EBP+OCH] , 204H ;правая кнопка
        JE     RBUTTON
        CMP    DWORD PTR  [EBP+0CH],111H ;WM_COMMAND
    JE     WMCOMMAND
    JMP    DEFWNDPROC
;пришло сообщение WM_COMMAND
;в частности, такое сообщение приходит и при выборе
;пункта меню
WMCOMMAND:
;здесь следует проверить, какой пункт меню выбран
;код пункта меню указан в файле ресурсов
;у нас этот код равен 101  (один пункт меню)
;этот код будет помещен в младшем слове параметра WPARAM
    CMP    WORD PTR  [EBP+10H],102     ;exit&
    JE     WMDESTROY
    CMP    WORD PTR  [EBP+10H],101     ;Пункт меню
    JNE    NO_IT
;выдать сообщение
    PUSH   O            ;MB_OK - тип сообщения
    PUSH   OFFSET TEXTMC  ;заголовок окна сообщения
    PUSH   OFFSET TEXTM   ;сообщение
    PUSH   DWORD PTR  [EBP+08H]    ;дескриптор окна
    CALL   MessageBoxA
NO_IT:
        JMP    DEFWNDPROC
;нажатие правой кнопки приводит к закрытию окна
RBUTTON:
        JMP     WMDESTROY
;нажатие левой кнопки вызывает звуковой сигнал
LBUTTON:
        CALL   BEE
        MOV    EAX,0
        JMP    FINISH
WMCREATE:
        MOV      EAX, .0
        JMP      FINISH
DEFWNDPROC:
        PUSH     DWORD PTR  [EBP+14H]
        PUSH     DWORD PTR  [EBP+10H]
        PUSH     DWORD PTR  [EBP+OCH]
```

```
        PUSH      DWORD PTR  [EBP+08H]
        CALL      DefWindowProcA
        JMP       FINISH
WMDESTROY:
        PUSH       0
        CALL      PostQuitMessage
        MOV       EAX,  0  ;функция возвращает 0
FINISH:
        POP   EDI
        POP   ESI
        POP   EBX
        POP   EBP
        RET   16
WNDPROC              ENDP
;---------------------------------------------------------------
;процедура короткого звукового сигнала
BEE  PROC
        MOV       ECX,100
        PUSH      ECX
_OP:
        PUSH      0
        PUSH      0
;библиотечная процедура, дающая короткий звуковой сигнал
        CALL      Beep
        POP       ECX
        DEC       ECX
        PUSH      ECX
        LOOPD     _OP
        POP  ECX
        RET
BEE  ENDP
_TEXT  ENDS
        END  START
```

Рис. 24.9. Пример использования файла ресурсов.

Одним из основных ресурсов является **Диалог.** Он представляет из себя диалоговое окно, которое может содержать другие органы управления (ресурсосоздаваемые с использованием визуальных средств в редакторе ресурсов). Диалог, как и обычное окно, перед тем как использоваться в программе, должен быть создан. Только для этого используется не функция создания окна (CreateWindow), а функция DialogBox. Как и обычное окно, диалоговое окно имеет свою функцию, обрабатывающую сообщения, приходящие на диалог. Структура этой функции точно такая же, как функции окна. Отличие заключается в том, что:

1. Функция возвращает 0, если данное сообщение в функции не обработано и 1 (не ноль), если сообщение обработано.
2. Некоторые сообщения окна заменены другими. Появились новые сообщения. Например, вместо сообщения WM_CREATE приходит сообщение WM_INITDIALOG. Основные же сообщения остались теми же.

Вообще говоря, при выполнении функции DialogBox автоматически создается функция обработки окна диалога. Функция же, которую мы определяем в программе, является, таким образом, вторичной и вызывается из первой функции. Необходимость такого построения очевидна. При создании диалога мы определяем его свойства и свойства других ресурсов (кнопок, окон редактирования, списков, переключателей и т.д.). Свойства должна поддерживать первая функция. Любое сообщение на окно диалога приходит сначала в первую функцию. Оттуда сразу вызывается вторая. Если вторая функция возвращает 0, то данное сообщение обрабатывает первая функция, если 1, то первая функция полагает, что обработку на себя взяла вторая.

Каждый ресурс, который мы поместим на диалоговое окно, имеет свой номер. В пределах данного диалога номер уникален. Как и в случае с обычным окном, события, происходящие с органом управления, отражаются в сообщении WM_COMMAND. В младшем слове параметра wParam будет содержаться идентификатор ресурса, в старшем - код события.

Ниже (Рис. 24.11) представлено простейшее использование диалога. В диалоговом окне имеется единственная кнопка, по нажатию которой происходит выход из программы. Создание диалога осуществляется вызовом функции DialogBoxParamA (все остальные функции, создающие диалог, работают через эту функцию). На Рис. 24.10 представлен файл ресурса.

```
#define IDC_BUTTON1 55  //идентификатор кнопки
DIAL1 DIALOG 0, 0, 240, 120  //положение диалога
STYLE DS_MODALFRAME | DS_3DLOOK | DS_CONTEXTHELP | WS_POPUP |
WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Окно диалога" //заголовок окна
FONT 8, "MS Sans Serif"
{
 CONTROL "OK", IDC_BUTTON1, "button", BS_PUSHBUTTON | BS_CENTER
| WS_CHILD | WS_VISIBLE | WS_TABSTOP, 176, 13, 50, 14
}
```

Рис. 24.10. Файл ресурса для программы на Рис. 24.11.

```
.386P
;плоская модель
;в стек справа налево
.MODEL FLAT

;внешние процедуры
EXTRN            DispatchMessageA:NEAR
EXTRN            ExitProcess:NEAR
```

```
EXTRN           GetModuleHandleA:NEAR
EXTRN           PostQuitMessage:NEAR
EXTRN           GetMessageA:NEAR
EXTRN           TranslateMessage:NEAR
EXTRN           DialogBoxParamA:NEAR
EXTRN           EndDialog:NEAR
PUBLIC  DLGPROC

;структуры
MSGSTRUCT  STRUC
MSHWND           DD    ?
MSMESSAGE        DD    ?
MSWPARAM         DD    ?
MSLPARAM         DD    ?
MSTIME           DD    ?
MSPT             DD    ?
MSGSTRUCT  ENDS
;---------
;сегмент данных
_DATA  SEGMENT  DWORD  PUBLIC  USE32  'DATA'
NEWHWND          DD 0
MSG              MSGSTRUCT   <?>
HINST            DD 0
DIAL             DB  'DIAL1',0 /имя - идентификатор диалога
_DATA  ENDS
;сегмент кода
_TEXT   SEGMENT  DWORD  PUBLIC  USE32  'CODE'
START:
      PUSH     0
      CALL     GetModuleHandleA
      MOV      [HINST], EAX
;создать диалог
      PUSH 0            ;возвращаемая функцией диалога величина
      PUSH OFFSET DLGPROC ;функция окна диалога
      PUSH 0        ;дескриптор окна, где появляется диалог
      PUSH OFFSET DIAL ;имя - идентификатор диалога
      PUSH [HINST]
      CALL DialogBoxParamA
;петля обработки сообщений
MSG_LOOP:
      PUSH     O
      PUSH     O
      PUSH     O
      PUSH     OFFSET MSG
```

```
        CALL    GetMessageA
        CMP     AX, 0
        JE      END_LOOP
        PUSH    OFFSET MSG
        CALL    TranslateMessage
        PUSH    OFFSET MSG
        CALL    DispatchMessageA
        JMP     MSG_LOOP
END_LOOP:
        PUSH    [MSG.MSWPARAM]
        CALL    ExitProcess
;------------------------------------------------------------
;процедура диалога
;расположение параметров в стеке
;как и в обычной функции окна
;  [BP+014H]        ;LPARAM
;  [BP+10H]         ;WAPARAM
;  [BP+OCH]         ;MES
;  [BP+8]           ;HWND
DLGPROC             PROC
        PUSH  EBP
        MOV   EBP,ESP
        PUSH  EBX
        PUSH  ESI
        PUSH  EDI
        MOV   EAX, 0
        CMP   DWORD PTR  [EBP+0CH],110H ;WM_INITDIALOG
        JE    WMINITDIALOG
        CMP   DWORD PTR  [EBP+0CH],111H ;WM_COMMAND
        JE    WMCOMMAND
        JMP   FINISH
;пришло сообщение WM_COMMAND
;в частности, такое сообщение приходит и при нажатии кнопки
WMCOMMAND:
        CMP   WORD PTR [EBP+10H],55 ; кнопка
        JNE    FINISH
;нажатие правой кнопки приводит к закрытию окна
        PUSH     O
        CALL     PostQuitMessage ;сообщение о выходе из про-
граммы
        PUSH O
        PUSH   DWORD PTR  [EBP+08H]
        CALL    EndDialog    ;закрыть диалоговое окно
       MOV    EAX,1
       JMP     FINISH
```

```
;здесь те команды, которые необходимо выполнить
;при инициализации диалога, у нас здесь ничего
WMINITDIALOG:
        JMP     FINISH
FINISH:
        POP   EDI
        POP   ESI
        POP   EBX
        POP   EBP
        RET   16
DLGPROC             ENDP
;--------      ---  ----     ------------   --  ---  ----
_TEXT ENDS
END START
```

Рис. 24.11. Простой пример использования диалогового окна.

Заключительные замечания.

Материал, изложенный в данной главе, базируется на системных вызовах операционной системы Windows (API функциях). Современные средства программирования под Windows всецело основываются на использование библиотек объектов. Большое распространение нашло визуальное программирование. Мы рассматривали программирование для Windows более низкого уровня, чем то, которое обычно сейчас используется большинством программистов. Знание особенностей системных вызовов поможет Вам программировать более грамотно и профессионально. Я рекомендовал начинать программировать для Windows, используя только системные вызовы.

Несколько слов следует сказать по поводу совместимости Windows 95 (98) и Windows NT 4.0. Обычно говорят о почти полной совместимости этих операционных систем на уровне API-функций. С этим довольно трудно согласиться. Существует по крайней мере три уровня несовместимости.

1. Полное отсутствие в той или иной операционной системе данной функции. И хотя таких функций немного, но они имеются.
2. Несовпадение некоторых значений входных параметров. Например, для функции CreateFile параметр dwShareMode может принимать значение FILE_SHARE_DELETE, которое разрешено только для Windows NT.
3. Имеется еще один уровень несовместимости, как правило, не отраженный в документации, но который может сказаться на работе программного обеспечения. Автор обнаружил эту несовместимость в работе некоторых сетевых функций. Например, согласно документации некоторый параметр может принимать значение как NULL, так и «пустая строка». Но одна из операционных систем, почему-то правильно работает только с одним значением параметра.

Глава 26. Программирование в защищенном режиме.

Без надобности носимый набрюшник - вреден.

Козьма Прутков.

Данная глава является логическим продолжением главы 5. В главе 5 явно не говорится о том, как программировать в защищенном режиме, хотя и дается представление о нем. Программы данной главы рассчитаны на MASM.EXE версии 5.0. При использовании MASM.EXE более высоких версий могут возникнуть проблемы следующего порядка:

1. Команда типа MOV AX,CR0 будет давать ошибку, ее следует поменять на MOV EAX,CR0.

2. Команда вида LIDT QWORD MET1 также не пройдет. Следует поменять ее на LIDT FWORD MET1. FWORD - шестибайтовая структура. Соответственно для резервирования такой структуры можно использовать DF.

Говоря о защищенном режиме, имейте в виду, что мы уже работали в нем с программами предыдущей главы. Однако в Windows все управление брала на себя операционная система. Здесь нам придется писать программы, берущие всю ответственность на себя. Фактически написать программу, которая самостоятельно работает в защищенном режиме, это написать маленькую операционную систему.

I.

Прежде всего вспомним некоторые положения, рассмотренные в главе 5.

1. В защищенном режиме физический адрес формируется из селектора и смещения. Селектор содержится в сегментном регистре и представляет собой не сегментный адрес, как в реальном режиме, а индекс-указатель на дескриптор в дескрипторной таблице. Дескриптор представляет собой описатель сегмента - физический адрес, размер сегмента, байт-описатель сегмента (байт доступа).

2. Система имеет одну глобальную дескрипторную таблицу и может иметь несколько локальных дескрипторных таблиц. В случае одной задачи локальные дескрипторные таблицы не используются.

3. Содержимое сегментного регистра в защищенном режиме имеет следующую структуру: первые три бита носят служебный характер (см. главу 5), оставшиеся 13 бит представляют собой индекс, другими словами номер дескриптора в дескрипторной таблице. Бит 2 определяет, какая дескрипторная таблица используется (0 - глобальная).

4. То, что под индекс, указывающий на дескриптор, отведены старшие 13 бит, далеко не случайный факт. Дело в том, что длина дескриптора 8 байт. Вначале идет нулевой, неиспользуемый дескриптор. Если DESC0 - метка начала нулевого дескриптора, DESC - метка начала другого дескриптора, то разность DESC-DESC0 - кратна 8 и представляет собой как раз индекс дескриптора DESC, помещенный в старшие 13 бит. Этот факт значительно упрощает работу с дескрипторами в программе.

5. Мы привыкли оперировать двухкомпонентным адресом, состоящим из сегмента и смещения. Покажем, как перевести его в физический адрес, который должен быть помещен в дескриптор. Пусть сегментный адрес помещен в AX, а смещение - в BX. Получим трехбайтный (24-битный) физический адрес в DL:AX. Следующие команды справляются с указанной задачей.

```
MOV DL,AH
SHL AX,4
SHR DL,4
ADD AX,BX
ADC DL,0
```

6. В защищенном режиме программные прерывания перестают работать. Поэтому нам придется писать собственные процедуры работы с внешними устройствами. Для реализации аппаратных прерываний необходимо создавать свою таблицу дескрипторов прерываний. Если программа не собирается долго находиться в защищенном режиме, то простейший выход из этого положения - это просто запретить все прерывания на время работы в защищенном режиме.

7. В дальнейшем Вам понадобятся сведения о CMOS - памяти, питающейся от независимого источника (см. главу 23). При запуске компьютера содержимое CMOS анализируется процедурами BIOS, которые извлекают оттуда информацию о конфигурации системы, а также текущую дату и время. Доступ к CMOS осуществляется через порты 70H и 71H.

```
Чтение из CMOS:
      MOV  AL,XXH
      OUT  70H,AL  ;выбор номер ячейки  CMOS
      JMP  $+2     ;задержка
      IN   AL,71H  ;ввод байта из CMOS
Запись в CMOS:
      MOV  AL,XXH
      OUT  70H,AL  ;выбор номер ячейки CMOS
      JMP  $+2     ;задержка
      MOV  AL,YYH  ;байт для ввода в CMOS
      OUT  71H,AL
```

Порт 70H служит не только для индексирования ячеек CMOS, но и для разрешения или запрещения NMI (немаскируемого прерывания). Если бит 7 равен 0, то NMI разрешается, в противном случае запрещается. Тут есть один интересный момент. Управление NMI и индексацию можно совместить в одной команде. Если в AL лежит байт с нулевым 7-м битом, то OUT 70H,AL не только индексирует ячейку CMOS, но и размаскирует NMI. Адреса ячеек CMOS находятся в промежутке 00H-3FH. Если адрес ячейки больше 3FH, то автоматически учитывается только младший байт.

8. Ячейка с номером FH CMOS называется байтом состояния перезагрузки. Этот байт считывается после сброса центрального процессора, чтобы определить, не был ли сброс вызван для вывода из защищенного режима. Если содержимое этого байта равно 5, то осуществляется выполнения команды

JMP FAR PTR [0:467H]

```
;работаем в защищенном режиме процессора
.286P
;байты доступа
; 10000000B - сегмент есть в памяти
;00011000B - сегмент кода
;00010000B - сегмент данных
; 00000100B - сегмент расширяется вниз
/00000100B - согласованный сегмент
;00000010B - разрешена запись
;сегмент данных
DATA_AC   EQU 10010010B
/сегмент кода
CODE_AC   EQU 10011100B
/сегмент стека
STACK_AC EQU 10010110B
DATA SEGMENT
BEG_DATA = $
;строка для вывода
MSG      DB 'HELLO! Я в защищенном режиме!',0
MSG1   DB 'Привет! Я снова в реальном режиме! ' ,0
/данные для вывода на экран
COLUMNS DB ?      /количество столбцов на экране
ROWS    DB ?      /количество строк на экране
SEL_BUF DW ?      ;селектор видеобуфера
SEGBUF  DW ?      ;сегментный адрес видеобуфера
;здесь хранятся сегментные регистры
_SS      DW ?
_DS      DW ?
_ES      DW ?
_SP      DW ?
;глобальная дескрипторная таблица
/нулевой  дескриптор
GDT0      DQ 0
/дескриптор для GDT
GDT_GDT   DQ 0
/дескриптор для сегмента данных
GDT_DS    DQ 0
```

```
;дескриптор для сегмента кода
GDT_CS    DQ 0
;дескриптор для сегмента стека
GDT_SS    DQ 0
;дескриптор для видеопамяти цветного дисплея
GDT_CRT   DQ 0
;дескриптор для видеопамяти монохромного дисплея
GDT_MDA   DQ 0
GDT_SIZE = $-GDTO    /размер глобальной дескрипторной таблицы
DSEG_SIZE  =  $-BEG_DATA /размер сегмента данных
DATA ENDS
;сегмент стека
ST1 SEGMENT STACK 'STACK'
      DB 100 DUP(?)
ST1 ENDS
;сегмент кода
CODE  SEGMENT
      ASSUME CS:CODE, DS:DATA, SS:ST1
BEG:
;инициализируем сегмент данных
      MOV AX,DATA
      MOV DS,AX
/определяем базовый адрес видеопамяти
      CALL WHAT_CRT
/подготовка перехода в защищенный режим
      CALL  INI_PROT
;переключение в защищенный режим
      CALL  SET_PROT
/стираем экран в защищенном режиме
      MOV   AH,77H
      CALL CLS
;вывод сообщения в защищенном режиме
;координаты
      MOV BX,0    /координата Y
      MOV AX,15   /координата X
;адрес выводимой строки
      MOV SI,OFFSET MSG
      MOV AH,41H  /атрибут
      CALL WRITE
;пауза
      CALL  PAUSE
/вернуться в реальный режим
      CALL  SET_REAL
```

```
;стираем экран в реальном режиме
      MOV AH,7
      CALL  CLS
;вывод сообщения в реальном режиме
;координаты
      MOV BX, 0    ;координата Y
      MOV AX,15    ;координата X
;адрес выводимой строки
      MOV SI,OFFSET MSG1
      MOV AH,13    ;атрибут
      CALL WRITE
;пауза
      CALL  PAUSE
;конец работы программы
      MOV AH,4CH
      INT 21H
;раздел процедур
;процедура задержки
PAUSE PROC
      PUSH CX
      MOV  CX,50
PL:
      PUSH CX
      MOV  CX,0FFFFH
PL1:
      LOOP PL1
      POP  CX
      LOOP PL
      POP  CX
      RETN
PAUSE ENDP
;определение параметров для видеобуфера
WHAT_CRT PROC
      MOV AX,40H
      MOV ES,AX
      MOV BX,ES:[4AH]
      MOV COLUMNS,BL
      MOV BL,ES:[84H]
      INC BL
      MOV ROWS,BL
      MOV BX,ES:[63H]
      CMP BX,3D4H
      JNE NO_COLOR
```

```
;цветной адаптер
        MOV SEL_BUF,(GDT_CRT-GDT0)
        MOV SEG_BUF,0B800H
        MOV ES,SEG_BUF
        JMP SHORT EXIT
NO_COLOR:
;адаптер MDA
        MOV SEL_BUF,(GDT_MDA-GDT0)
        MOV SEG_BUF,0B000H
        MOV ES,SEG_BUF
EXIT:
        RETN
WHAT_CRT ENDP
;процедура очистки экрана
;в ES находится либо селектор, либо сегмент
;в AH - атрибут
CLS   PROC
        MOV  CX,2000
        MOV  AL, 32
        XOR  DI,DI
L2:
        STOSW
        LOOP L2
        RETN
CLS   ENDP
;процедура выводит сообщение
;в ES либо селектор экрана, либо адрес сегмента
;AX,BX - X,Y
;SI - адрес строки
;AH - атрибут
WRITE  PROC
;расчет смещения в видеопамяти
        PUSH AX
        MOV DL,COLUMNS
        MUL DL
        ADD AX,BX
        SHL AX,1
        MOV DI,AX
        POP AX
L01:
        LODSB
        CMP AL,0
        JZ  EN
        STOSW
        JMP SHORT L01
```

```
EN:
      RETN
WRITE ENDP
;процедура инициализации защищенного режима
INI_PROT PROC
;заполнение GDT
;———— вычисление физического адреса————
      MOV AX,DATA
      MOV DL,AH
      SHR DL,4
      SHL AX,4
      MOV SI,AX
      MOV DI,DX
;———— заполнение дескриптора GDT————
      ADD AX,OFFSET GDT0
      ADC  DL,0
      MOV BX,OFFSET GDT_GDT
      MOV WORD PTR [BX],GDT_SIZE-1
      MOV  [BX+2],AX
      MOV  [BX+4],DL
      MOV BYTE PTR [BX+5],DATA_AC
;———— заполнение дескриптора для сегмента данных————
      MOV BX,OFFSET GDT_DS
      MOV AX,SI
      MOV DX,DI
      MOV WORD PTR [BX],DSEG_SIZE-1
      MOV  [BX+2],AX
      MOV  [BX+4],DL
      MOV BYTE PTR [BX+5],DATA_AC
;———— заполнение дескриптора для сегмента кода————
      MOV BX,OFFSET GDT_CS
      MOV AX,CS
      MOV DL,AH
      SHR DL,4
      SHL AX,4
      MOV WORD PTR [BX],CSEG_SIZE-1
      MOV  [BX+2],AX
      MOV  [BX+4],DL
      MOV BYTE PTR [BX+5],CODE_AC
;———— заполнение дескриптора для сегмента стека————
      MOV BX,OFFSET GDT_SS
      MOV AX,SS
      MOV DL,AH
      SHR DL,4
```

```
        SHL AX,4
        MOV WORD PTR  [BX],400-1
        MOV  [BX+2],AX
        MOV  [BX+4],DL
        MOV BYTE PTR  [BX+5],DATA_AC
;---------заполнение дескриптора для видеопамяти CRT-----------
        MOV BX,OFFSET GDT_CRT
        MOV WORD PTR  [BX],3999
        MOV WORD PTR  [BX+2],8000H
        MOV BYTE PTR  [BX+4],OBH
        MOV BYTE PTR  [BX+5],DATA_AC
;---------заполнение дескриптора для видеопамяти MDA-----------
        MOV BX,OFFSET GDT_MDA
        MOV WORD PTR  [BX],3999
        MOV BYTE PTR  [BX+4],0BH
        MOV BYTE PTR  [BX+5],DATA_AC
;---------адрес возврата из защищенного режима-----------------
        PUSH DS
        MOV  AX,40H
        MOV  DS, AX
        MOV  word ptr DS: [67H],OFFSET SHUT_DOWN_RETURN
        MOV  DS:[69H],CS
        POP  DS
;---------маскируем все прерывания-----------------------------
;записываем в ячейку OFH CMOS код 5
;чтобы обеспечить возврат после сброса процессора
;одновременно маскируем NMI
        CLI
        MOV  AL,8FH
        OUT  70H,AL
        JMP  NEXT1        ;небольшая задержка
NEXT1:
        MOV  AL,5
        OUT  71H,AL
        RETN
INI_PROT  ENDP
;процедура переключения в защищенный режим
SET_PROT  PROC
        MOV   _SS,SS
        MOV   _ES,ES
;загружаем регистр GDTR
        LGDT GDT_GDT
; устанавливаем защищенный режим
        MOV  AX, 1
        LMSW AX
```

```
;·--——JMP FAR FLUSH, для сбрасывания очереди команд
        DB    0EAH
        DW    OFFSET FLUSH
        DW     (GDT_CS-GDT0)
FLUSH:
;загрузка селекторов
        MOV  AX, (GDT_DS-GDTO)
        MOV   DS,AX
        MOV  AX,SEL_BUF
        MOV  ES,AX
        MOV   AX,(GDT_SS-GDT0)
        MOV   SS,AX
        RETN
SET_PROT  ENDP
;процедура возврата в реальный режим
SET_REAL  PROC
;запомнить содержимое регистра стека
        MOV _SP,SP
;сброс процессора
        MOV AL,0FEH
        OUT  64H,AL
;ожидание возврата по метке SHUT_DOWN_RETURN
_WAIT:
        JMP _WAIT
;--теперь мы в реальном режиме--
;метка возврата
SHUT_DOWN_RETURN:
;восстанавливаем значения всех сегментных регистров
        MOV AX,DATA
        MOV DS,AX
        MOV  SS,_SS
        MOV  SP,_SP
        MOV ES,_ES
/разрешаем все прерывания
;немаскируемые прерывания
        MOV   AX,0DH
        OUT   70H,AL
;маскируемые прерывания на уровне контроллера прерываний
        XOR   AL,AL
        OUT   21H,AL
/разрешаем маскируемые прерывания на уровне микропроцессора
        STI
        RETN
SET_REAL ENDP
```

```
CSEG_SIZE = $-BEG
CODE ENDS
        END BEG
```

Рис. 20.1. Вывод в защищенном режиме строки и очистка экрана.

На Рис. 20.1 представлена программа, работающая в защищенном режиме. Программа достаточно хорошо прокомментирована, поэтому рассмотрим лишь несколько замечаний.

1. Программа переходит в защищенный режим. Стирает экран и печатает строку. Затем происходит возврат в реальный режим и снова печать строки.

2. Поскольку в защищенном режиме программа должна находиться недолго, мы просто отключаем все прерывания.

3. Обращаю Ваше внимание, что процедуры печати строки и очистки экрана одни и те же в реальном и защищенном режиме. Но если в реальном режиме в сегментных регистрах должны находиться адреса сегментов, в защищенном режиме в сегментных регистрах должны содержаться индексы, указывающие на дескрипторы сегментов.

4. Как отмечалось раньше, если в защищенном режиме работает всего одна программа, нет смысла создавать локальную дескрипторную таблицу. Поэтому моя программа использует лишь глобальную дескрипторную таблицу.

Одна из преимуществ защищенного режима - возможность использовать всю память компьютера. На Рис. 20.2 представлена программа, аналогичная предыдущей, но выполняющая еще одну функцию. Она копирует экранную область в память за 1Мб. Чтобы использовать эту память, предварительно следует открыть адресную линию А20, что делает процедура ENABLE. Заметим, что аналогичную операцию делала программа на Рис. 5.2 из главы 5. Однако там было использовано для этой цели прерывание 15Н.

```
;работаем в защищенном режиме процессора
.286P
;байты доступа
;10000000B - сегмент есть в памяти
;00011000B - сегмент кода
;00010000B - сегмент данных
;00000100B - сегмент расширяется вниз
;00000100B - согласованный сегмент
;00000010B - разрешена запись
;сегмент данных
DATA_AC   EQU 10010010B
;сегмент кода
CODE_AC   EQU 10011100B
;сегмент стека
STACK_AC EQU 10010110B
DATA SEGMENT
BEG_DATA = $
```

```
;строка для вывода
MSG      DB 'HELLO! Я в защищенном режиме!',0
MSG1     DB 'Привет! Я снова в реальном режиме!',0
MSG2     DB 'Копия экрана отправлена в память',0
;данные для вывода на экран
COLUMNS DB ?       ;количество столбцов на экране
ROWS     DB ?       ;количество строк на экране
SEL_BUF DW ?       ;селектор видеобуфера
SEG_BUF DW ?       ;сегментный адрес видеобуфера
;здесь хранятся сегментные регистры
_SS      DW ?
_DS      DW ?
_ES      DW ?
_SP      DW ?
;глобальная дескрипторная таблица
;нулевой дескриптор
GDT0      DQ 0
;дескриптор для GDT
GDT_GDT  DQ 0
;дескриптор для сегмента данных
GDT_DS   DQ 0
;дескриптор для сегмента кода
GDT_CS   DQ 0
;дескриптор для сегмента стека
GDT_SS   DQ 0
;дескриптор для видеопамяти цветного дисплея
GDT_CRT  DQ 0
; дескриптор для видеопамяти монохромного дисплея
GDT_MDA  DQ 0
;дескриптор расширенной памяти, куда будем копировать экран
GDT_MEM  DQ 0
GDT_SIZE = $-GDT0    ;размер глобальной дескрипторной таблицы
DSEG_SIZE = $-BEG_DATA ;размер сегмента данных
DATA ENDS
;сегмент стека
ST1 SEGMENT STACK 'STACK'
      DB 100 DUP(?)
ST1 ENDS
;сегмент кода
CODE SEGMENT
      ASSUME CS:CODE, DS:DATA, SS:ST1
BEG:
;инициализируем сегмент данных
      MOV AX,DATA
      MOV DS,AX
```

```
;определяем базовый адрес видеопамяти
      CALL WHAT_CRT
;подготовка перехода в защищенный режим
      CALL INI_PROT
;переключение в защищенный режим
      CALL SET_PROT
;стираем экран в защищенном режиме
      MOV  AH,77H
      CALL CLS
;вывод сообщения в защищенном режиме
;координаты
      MOV BX,0    ;координата Y
      MOV AX,15   ;координата X
;адрес выводимой строки
      MOV SI,OFFSET MSG
      MOV AH,41H   /атрибут
      CALL WRITE
;пауза
      CALL PAUSE
;копируем экран в память
      CALL TO_MEM
;стираем экран
      MOV AH,39H
      CALL CLS
;вывод сообщения в защищенном режиме
      MOV BX,0    /координата Y
      MOV AX,15   /координата X
      MOV SI,OFFSET MSG2
      MOV AH,71H   ;атрибут
      CALL WRITE
;пауза
      CALL PAUSE
/копируем экран из памяти
      CALL FROM_MEM
;пауза
      CALL PAUSE
/вернуться в реальный режим
      CALL SET_REAL
;стираем экран в реальном режиме
      MOV AH,7
      CALL CLS
/вывод сообщения в реальном режиме координаты
      MOV BX,0    /координата Y
      MOV AX,15   ;координата X
```

```
;адрес выводимой строки
        MOV SI,OFFSET MSG1
        MOV AH,13  ;атрибут
        CALL  WRITE
;пауза
        CALL  PAUSE
;конец работы программы
        MOV AH,4CH
        INT  21H
;раздел процедур процедура задержки
;для быстрого компьютера следует увеличить значение, засылаемое в CX
PAUSE  PROC
        PUSH  CX
        MOV  CX,50
PL:
        PUSH  CX
        MOV   CX,0FFFFH
PL1:
        LOOP   PL1
        POP    CX
        LOOP   PL
        POP    CX
        RETN
PAUSE  ENDP
;определение параметров для видеобуфера
WHAT_CRT  PROC
        MOV AX,40H
        MOV ES,AX
        MOV BX,ES:[4AH]
        MOV COLUMNS,BL
        MOV BL,ES:[84H]
        INC BL
        MOV ROWS,BL
        MOV BX,ES:[63H]
        CMP BX,3D4H
        JNE  NO_COLOR
        MOV  SEL_BUF,(GDT_CRT-GDT0)
        MOV SEG_BUF,0B800H
        MOV ES,SEG_BUF
        JMP  SHORT  EXIT
NO_COLOR:
;адаптер MDA
        MOV  SEL_BUF,(GDT_MDA-GDT0)
        MOV SEG_BUF,0B000H
        MOV ES,SEG_BUF
```

```
EXIT:
      RETN
WHAT_CRT ENDP
;процедура очистки экрана
;в ES находится либо селектор, либо сегмент
;в AH - атрибут
CLS   PROC
      MOV  CX,2000
      MOV  AL,32
      XOR  DI,DI
L2:
      STOSW
      LOOP L2
      RETN
CLS   ENDP
;процедура выводит сообщение
;в ES либо селектор экрана, либо сегмент
;AX,BX - X,Y
;SI - адрес строки
;AH - атрибут
WRITE PROC
;расчет смещения в видеопамяти
      PUSH AX
      MOV  DL,COLUMNS
      MUL  DL
      ADD  AX,BX
      SHL  AX,1
      MOV  DI,AX
      POP  AX
L01:
      LODSB
      CMP  AL,0
      JZ   EN
      STOSW
      JMP  SHORT L01
EN:
      RETN
WRITE ENDP
;процедура инициализации защищенного режима
INI_PROT PROC
; заполнение GDT
;----------------
      MOV  AX,DATA
      MOV  DL,AH
      SHR  DL,4
```

```
        SHL  AX,4
        MOV  SI,AX
        MOV  DI,DX
;-----------------
        ADD  AX,OFFSET GDT0
        ADC  DL,0
        MOV  BX,OFFSET GDT_GDT
        MOV  WORD PTR [BX],GDT_SIZE-1
        MOV   [BX+2],AX
        MOV   [BX+4],DL
        MOV  BYTE PTR [BX+5],DATA_AC
;-----------------
        MOV  BX,OFFSET GDT_DS
        MOV  AX,SI
        MOV  DX,DI
        MOV  WORD PTR [BX],DSEG_SIZE-1
        MOV   [BX+2],AX
        MOV   [BX+4],DL
        MOV  BYTE PTR [BX+5],DATA_AC
;-----------------
        MOV  BX,OFFSET GDT_CS
        MOV  AX,CS
        MOV  DL,AH
        SHR  DL,4
        SHL  AX,4
        MOV  WORD PTR [BX],CSEG_SIZE-1
        MOV   [BX+2],AX
        MOV   [BX+4],DL
        MOV  BYTE PTR [BX+5],CODE_AC
;-----------------
        MOV  BX,OFFSET GDT_SS
        MOV  AX,SS
        MOV  DL,AH
        SHR  DL,4
        SHL  AX,4
        MOV  WORD PTR [BX],400-1
        MOV   [BX+2],AX
        MOV   [BX+4],DL
        MOV  BYTE PTR [BX+5],DATA_AC
;-----------------
        MOV  BX,OFFSET GDT_CRT
        MOV  WORD PTR [BX],3999
        MOV  WORD PTR [BX+2],8000H
        MOV  BYTE PTR [BX+4],0BH
        MOV  BYTE PTR [BX+5],DATA_AC
```

```
;-----------------
        MOV BX,OFFSET GDT_MDA
        MOV WORD PTR  [BX],3999
        MOV BYTE PTR  [BX+4],0BH
        MOV BYTE PTR  [BX+5],DATA_AC
;-------дескриптор области озу для копирования экрана-------
        MOV BX,OFFSET GDT_MEM
        MOV WORD PTR  [BX],3999
        MOV WORD PTR  [BX+2],0000H
        MOV BYTE PTR  [BX+4],15H
        MOV BYTE PTR  [BX+5],DATA_AC
;-------------------адрес возврата из защищенного режима
        PUSH DS
        MOV  AX,40H
        MOV  DS,AX
        MOV  word ptr DS: [67H],OFFSET SHUT_DOWN_RETURN
        MOV  DS: [69H],CS
        POP  DS
;-----------------маскируем все прерывания
;записываем в ячейку OFH CMOS код 5
;чтобы обеспечить возврат после сброса процессора
        CLI
        MOV AL,8FH
        OUT 70H,AL
        JMP NEXT1
NEXT1:
        MOV AL,5
        OUT 71H,AL
        RETN
INI_PROT ENDP
;процедура переключения в защищенный режим
SET_PROT PROC
        MOV  _SS,SS
        MOV  _ES,ES
        CALL ENABLE
;загружаем регистр GDTR
        LGDT GDT_GDT
;устанавливаем защищенный режим
        MOV  AX, 1
        LMSW AX
;-------JMP FAR FLUSH, для сбрасывания очереди команд
        DB   OEAH
        DW   OFFSET FLUSH
        DW    (GDT_CS-GDTO)
FLUSH:
```

```
;загрузка селекторов
      MOV  AX,(GDT_DS-GDT0)
      MOV  DS,AX
      MOV AX,SEL_BUF
      MOV ES,AX
      MOV  AX,(GDT_SS-GDT0)
      MOV  SS,AX
      RETN
SET_PROT ENDP
;процедура возврата в реальный режим
SET_REAL PROC
      CALL DISABLE
;запомнить содержимое регистра стека
      MOV _SP,SP
;сброс процессора
      MOV AL,OFEH
      OUT 64H,AL
;ожидание возврата по метке SHUT_DOWN_RETURN
_WAIT:
      JMP _WAIT
;--теперь мы в реальном режиме--
;метка возврата
SHUT_DOWN_RETURN:
;восстанавливаем значения всех сегментных регистров
      MOV AX,DATA
      MOV DS,AX
      MOV  SS,_SS
      MOV  SP,_SP
      MOV ES,_ES
;разрешаем все прерывания
;немаскируемые прерывания
      MOV  AX,0DH
      OUT  70H,AL
;маскируемые прерывания на уровне контроллера прерываний
      XOR  AL,AL
      OUT  21H,AL
;маскируемые прерывания на уровне микропроцессора
      STI
      RETN
SET_REAL ENDP
;процедура копирования экрана в память
;в ES должен быть загружен селектор экран
TO_MEM PROC
      PUSH DS
      MOV  AX,(GDT_MEM-GDTO)
```

```
        MOV   DS,AX
        XOR   SI,SI
        XOR   DI,DI
        MOV   CX,4000
LL1:
        MOV   AL,ES:[SI]
        MOV    [DI],AL
        INC   SI
        INC   DI
        LOOP  LL1
        POP   DS
        RETN
TO_MEM ENDP
;процедура копирования из памяти на экран
;в ES должен быть загружен селектор экран
FROM_MEM  PROC
        PUSH  DS
        MOV   AX,(GDT_MEM-GDT0)
        MOV   DS,AX
        XOR   SI,SI
        XOR   DI,DI
        MOV   CX,4000
LL2:
        MOV   AL, [SI]
        MOV   ES:[DI],AL
        INC   SI
        INC   DI
        LOOP  LL2
        POP   DS
        RETN
FROM_MEM  ENDP
;открыть адресную шину A20
;управление линией A20 осуществляется через порты клавиатуры
ENABLE  PROC  NEAR
        MOV   AL,0D1H     ;команда управления линией A20
        OUT   64H,AL      ;порт состояния клавиатуры
        MOV   AL,ODFH     ;открыть A20
        OUT   60H,AL      ;клавиатурный порт
        RET
ENABLE  ENDP
;процедура закрывает адресную шину A20
DISABLE  PROC  NEAR
        MOV AL,OD1H
        OUT  64H,AL
```

```
        MOVAL,0DDH      ;закрыть A20
        OUT 60H,AL
        RET
DISABLE ENDP
CSEG_SIZE = $-BEG
CODE ENDS
        END BEG
```

Рис. 20.2. Программа, работающая в защищенном режиме: копирует содержимое экрана в адресное пространство за 1 Мб.

II.

До сих пор при входе в защищенный режим мы отключали все прерывания. Это наиболее простой способ написания программ. Сейчас предстоит рассмотреть вопрос об использовании прерываний в защищенном режиме.

Кратко суть проблемы. В защищенном режиме все прерывания делятся на обычные прерывания и исключения (особые случаи). Программные прерывания инициируются командой INT. Аппаратные прерывания происходят при возникновении внешнего события (нажатие клавиши и т.п.). При возникновении аппаратного прерывания флаг разрешения прерываний сбрасывается, и прерывания запрещаются. Исключения побуждаются ошибками, которые вызваны какими-либо командами. Типичной ошибкой является попытка записи в сегмент кода. При возникновении исключения прерывания не запрещаются.

Обработка прерываний в защищенном режиме базируется на таблице прерываний, которая содержит дескрипторы прерываний. Эти дескрипторы указывают на процедуру, которая должна выполниться при возникновении данного прерывания. Эти дескрипторы называются вентили. Положение таблицы определяется содержимым регистра IDTR. Загрузка регистра осуществляется командой LIDT.

Для обработки исключений зарезервирован **31-й** номер прерываний.
OOH ошибка при выполнении команды деления.
01H прерывания для пошаговой работы.
02H немаскируемое прерывание.
O3H прерывание по точке останова.
04H генерируется командой INTO.
05H генерируется командой BOUND, если проверяемое значение вышло за пределы заданного диапазона.
06H недействительный код команды.
07H отсутствие арифметического сопроцессора.
08H двойная ошибка - если при обработке исключения возникло еще одно.
09H превышение сегмента арифметическим сопроцессором.
OAH недействительный сегмент состояния задачи TSS.
OBH отсутствие сегмента, который может находиться в данный момент на диске.

OCH исключение при работе со стеком, например, переполнение.

ODH исключение по защите памяти, например, попытка доступа к сегменту при недостаточности привилегий.

OEH отказ страницы для процессоров 386 и выше.

OFH зарезервировано.

1OH исключение сопроцессора.

11H-1AH зарезервировано.

Заметим, что исключение можно вызвать и непосредственно командой INT. Перед тем как передать управление обработчику исключений, для многих зарезервированных прерываний процессор помещает в стек 16-битный код ошибки. Ниже приведена структура кода ошибки.

О - устанавливается в том случае, если ошибка произошла по внешним относительно выполняемой программы причинам.

1-2 - выбор локальной или глобальной таблицы - если бит 1 равен нулю, то бит 2 выбирает таблицу дескрипторов (0 - глобальная, 1 - локальная).

3-15 - индекс дескриптора, при обращении к которому произошла ошибка.

Код ошибки помещается в стек в случае исключений: 08H, OAH, OBH, OCH, ODH.

Все исключения, кроме 01H, 08H, 09H, ODH, 1OH, обладают свойством повторной запускаемости. Это означает следующее: для таких исключений (кроме 3 и 4) в стек включается адрес не следующей команды, а прерванной. Если выполнить команду IRET, то снова будет выполнена прерванная команда. Если, например, сегмент, к которому было обращение, окажется отсутствующим в памяти, то возникшее исключение может вызвать процедуру подкачки сегмента с диска и снова выполнить прерванную команду.

Если мы посмотрим на таблицу исключений, то заметим, что исключения 0-7 соответствуют прерываниям реального режима. Эти прерывания вызваны внутренним состоянием микропроцессора. Однако для обработки аппаратных прерываний IRQO-IRQ7 используются номера прерываний от 08H до OFH (см. главу 9). Но в защищенном режиме эти номера зарезервированы для исключений. Возникающая проблема решается перепрограммированием контроллера прерываний, что и делается в приведенной ниже программе (процедура PROG_INT). Алгоритм перепрограммирования приведен в главе 9.

```
;работаем в защищенном режиме процессора
.286P
;байты доступа
;10000000B - сегмент есть в памяти
;00011000B - сегмент кода
;00010000B - сегмент данных
;00000100B - сегмент расширяется вниз
;00000100B - согласованный сегмент
;00000010B - разрешена запись
;00000110B - вентиль прерывания
```

```
;00000111B - вентиль исключения
;сегмент данных
DATA_AC   EQU 10010010B
;сегмент кода
CODE_AC   EQU 10011100B
;сегмент стека
STACK_AC EQU 10010110B
;сегмент таблицы IDT
IDT_AC    EQU 10010010B
;байт доступа вентиля прерывания
INT_AC    EQU 10000110B
;байт доступа вентиля исключения
TRAP_AC   EQU 10000111B
;сегмент стека
ST1 SEGMENT STACK 'STACK'
      DB 100 DUP(?)
ST1 ENDS
;сегмент данных
DATA SEGMENT
BEG_DATA=$
MSG DB 'Мы в защищенном режиме.', 0
_DL DB 0
;данные для вывода на экран
COLUMNS DB ?          ;количество столбцов на экране
ROWS    DB ?          ;количество строк на экране
SEL_BUF DW ?          ;селектор видеобуфера
SEG_BUF DW ?          ;сегментный адрес видеобуфера
;для хранения сегментных регистров
_SS DW ?
_ES DW ?
_SP DW ?
;переменная для таймера
TIME DW ?
;глобальная дескрипторная таблица
;нулевой дескриптор
GDT0      DQ 0
/дескриптор для GDT
GDT_GDT   DQ 0
/дескриптор для сегмента кода
GDT_CS    DQ 0
;дескриптор для сегмента данных
GDT_DS    DQ 0
/дескриптор для сегмента стека
GDT_SS    DQ 0
```

```
;дескриптор для видеопамяти цветного дисплея
GDT_CRT  DQ O
;дескриптор для видеопамяти монохромного дисплея
GDT_MDA  DQ O
;дескриптор таблицы прерываний
GDT_IDT  DQ O
GDT_SIZE = $-GDTO   /размер глобальной дескрипторной таблицы
;таблица прерываний
/вентили исключений
IDT_BEG=$
EXC_00    DQ 0
EXC_01    DQ 0
EXC_02    DQ 0
EXC_03    DQ 0
EXC_04    DQ 0
EXC_05    DQ 0
EXC_06    DQ 0
EXC_07    DQ 0
EXC_08    DQ 0
EXC_09    DQ 0
EXC_OA    DQ 0
EXC_OB    DQ 0
EXC_0C    DQ 0 .
EXC_OD    DQ 0
EXC_OE    DQ 0
EXC_OF    DQ 0
EXC_10    DQ 0
EXC_11    DQ 0
EXC_12    DQ 0
EXC_13    DQ 0
EXC_14    DQ 0
EXC_15    DQ 0
EXC_16    DQ 0
EXC_17    DQ 0
EXC_18    DQ 0
EXC_19    DQ 0
EXC_1A    DQ 0
EXC_1B    DQ 0
EXC_1C    DQ 0
EXC_1D    DQ 0
EXC_1E    DQ 0
EXC_1F    DQ 0
;вентили прерываний
INT_20   DQ 0 ; таймер
INT_21   DQ 0 ; клавиатура
```

```
INT_22   DQ 0
INT_23   DQ 0
INT_24   DQ 0
INT_25   DQ 0
INT_26   DQ 0
INT_27   DQ 0
INT_28   DQ 0
INT_29   DQ 0
INT_2A   DQ 0
INT_2B   DQ 0
INT_2C   DQ 0
INT_2D   DQ 0
INT_2E   DQ 0
INT_2F   DQ 0
IDT_SIZE=$-IDT_BEG    ;размер таблицы прерываний
;-----------------------------
DSEG_SIZE=$-BEG_DATA ;размер сегмента данных
DATA ENDS
;сегмент кода
CODE SEGMENT
      ASSUME CS:CODE, DS:DATA, SS:ST1
BEGIN:
      MOV AX,DATA
      MOV DS,AX
      CALL WHAT_CRT
;инициализировать дескрипторные таблицы
      CALL INI_PROT
;выйти в защищенный режим
      CALL SET_PROT
;координаты
      MOV BX,0   /координата Y
      MOV AX,15  /координата X
;адрес выводимой строки
      MOV SI,OFFSET MSG
      MOV AH,41H  ;атрибут
      CALL WRITE
;здесь можно записать какую-либо команду, вызывающую ошибку
/здесь цикл, выйти из которого можно только по ошибке либо
/по нажатии клавиши ESC
_WAI: JMP SHORT _WAI
;раздел процедур
/установка защищенного режима
SET_PROT PROC NEAR
      MOV _SS,SS
      MOV _ES,ES
```

```
        CALL  ENABLE
        CALL  PROG_INT
;загрузить регистр прерываний
        LIDT QWORD PTR GDT_IDT
;загрузить регистр глобальной дескрипторной таблицы
        LGDT QWORD PTR GDT_GDT
;устанавливаем защищенный режим
        MOV ' AX,1
        LMSW AX
;———————JMP  FAR  FLUSH,  для  сбрасывания  очереди  команд
        DB    OEAH
        DW    OFFSET  FLUSH
        DW     (GDT_CS-GDTO)
FLUSH:
;загрузка  селекторов
        MOV   AX,(GDT_DS-GDT0)
        MOV   DS,AX
        MOV AX,SEL_BUF
        MOV ES,AX
        MOV   AX,(GDT_SS-GDT0)
        MOV   SS,AX
;размаскировать прерывания на уровне контроллера
        XOR   AL,AL
        OUT   21H,AL
        OUT   OA1H,AL
        XOR   DL,DL
;размаскировать прерывания на уровне микропроцессора
        STI
        RETN
SET_PROT ENDP
;разрешение линии A20
ENABLE PROC NEAR
        MOV   AL,0D1H     ;команда управления линией A20
        OUT   64H,AL      ;порт состояния клавиатуры
        MOV   AL,ODFH     ;открыть A20
        OUT   60H,AL      ;клавиатурный порт
        RETN
ENABLE ENDP
;запрещение линии A20
DISABLE PROC NEAR
        MOV AL,0D1H
        OUT 64H,AL
        MOV AL,0DDH       ;закрыть A20
        OUT 60H,AL
        RETN
```

```
DISABLE  ENDP
;звуковой сигнал
;процедура взята из главы 6 (Рис. 6.5)
SOUND PROC NEAR
      PUSH AX
      PUSH CX
      MOV AL,10110110B              ;установка режима записи
      OUT 43H,AL
      IN  AL,61H
      OR  AL,3   .                  ;разрешить связь с таймером
      OUT 61H,AL
      MOV AX,1200
;установить частоту звука
      OUT 42H,AL
      MOV AL,AH
      OUT 42H,AL
      MOV CX,0FFFFH
;задержка
LOO: LOOP LOO
      MOV CX,0FFFFH
;задержка
LOO1: LOOP L001
;отключить канал от динамика, т.е. прекратить звук
      IN  AL,61H
      AND AL,11111100B
      OUT 61H,AL
      POP CX
      POP AX
      RET
SOUND ENDP
;перепрограммирование контроллера прерываний
PROG_INT PROC NEAR
;первый контроллер
;прерывания 0-7 получают номера 20H-27H
      MOV AH,20H
      MOV AL,11H
      OUT 20H,AL
      JMP SHORT $+2
      MOV AL,AH
      OUT 21H,AL
      JMP SHORT $+2
      MOV AL,4
      OUT 21H,AL
      JMP SHORT $+2
```

```
        MOV AL,1
        OUT 21H,AL
        JMP  SHORT  $+2
        MOV  AL,OFFH
        OUT 21H,AL
        JMP  SHORT  $+2
;второй контроллер
;прерывания 8-15 получают номера 28H-2FH
        MOV AH,28H
        MOV AL,11H
        OUT  OAOH,AL
        JMP  SHORT  $+2
        MOV AL,AH
        OUT 0A1H,AL
        JMP  SHORT  $+2
        MOV AL,4
        OUT  0A1H,AL
        JMP  SHORT  $+2
        MOV AL,1
        OUT  0A1H,AL
        JMP  SHORT  $+2
        MOV  AL,OFFH
        OUT  OA1H,AL
        JMP  SHORT  $+2
        RETN
PROG_INT  ENDP
;определение параметров видеосистемы
WHAT_CRT PROC NEAR
        MOV AX,40H
        MOV ES,AX
        MOV BX,ES:[4AH]
        MOV COLUMNS,BL
        MOV BL,ES:[84H]
        INC BL
        MOV ROWS,BL
        MOV BX,ES:[63H]
        CMP BX,3D4H
        JNE NO_COLOR
;цветной адаптер
        MOV- SEL_BUF,(GDT_CRT-GDT0)
        MOV SEG_BUF,0B800H
        MOV ES,SEG_BUF
        JMP SHORT EXIT
NO_COLOR:
```

```
;адаптер MDA
        MOV SEL_BUF,(GDT_MDA-GDT0)
        MOV SEG_BUF,0B000H
        MOV ES,SEG_BUF
EXIT:
        RETN
WHAT_CRT ENDP
;вывод на экран строки
;в ES либо селектор экрана, либо сегмент
;AX,BX - X,Y
;SI - адрес строки
;AH - атрибут
WRITE -PROC NEAR
;расчет смещения в видеопамяти
        PUSH AX
        MOV DL,COLUMNS
        MUL DL
        ADD AX,BX
        SHL AX,1
        MOV DI,AX
        POP AX
L01:
        LODSB
        CMP AL, 0
        JZ  EN
        STOSW
        JMP SHORT L01
EN:
        RETN
WRITE ENDP
;обработка клавиатурного прерывания
END_INT PROC NEAR
        IN   AL,61H
        MOV  AH,AL
        OR   AL,80H
        OUT  61H,AL
        XCHG AH,AL
        OUT  61H,AL
        MOV  AL,20H
        OUT  20H,AL
        RETN
END_INT ENDP
KEY PROC NEAR
        PUSH AX
        CALL SOUND
```

```
        IN    AL,60H
        CMP   AL,1
        JNZ   NO_ESC
        CALL  END_INT
        MOV   _DL,48
        CALL  SET_REAL   ;выход из защищенного режима
NO_ESC:
;————————————————конец прерывания
        CALL  END_INT
        POP   AX
        IRET
KEY ENDP
;обработка прерывания таймера
;вызывает звуковой сигнал, примерно раз в секунду
TIMER PROC NEAR
        CLI
        PUSH  AX
        MOV   AX,TIME
        CMP   AX,18
        JNZ   DAL
        MOV   TIME,0
        CALL  SOUND
        JMP   SHORT DAL1
DAL:
        INC   TIME
DAL1:
        MOV   AL,20H
        OUT   20H,AL
        POP   AX
        IRET
TIMER ENDP
;обработчик прерываний от первого контроллера
INT1 PROC NEAR
        PUSH  AX
        MOV   AL,20H
        OUT   20H,AL
        POP   AX
        MOV   _DL,50
        IRET
INT1 ENDP
;обработчик прерываний от второго контроллера
INT2 PROC NEAR
        PUSH  AX
        MOV   AL,20H
```

```
      OUT   20H,AL
      OUT   0AH,AL
      POP   AX
      MOV   _DL,51
      IRET
INT2 ENDP
;обработка исключений
EX00 PROC NEAR
      MOV   _DL,52  ;1-е исключение
      CALL  SOUND
      CALL  SOUND
      JMP   SET_REAL
EX00 ENDP
EX01 PROC NEAR
      MOV   _DL,53  ;1-е исключение
      CALL  SOUND
      CALL  SOUND
      JMP   SET_REAL
EX01 ENDP
EX02 PROC NEAR
      MOV   _DL,54  ;2-е исключение
      CALL  SOUND
      CALL  SOUND
      JMP   SET_REAL
EX02 ENDP
EX03 PROC NEAR
      MOV   _DL,55  ;3-е исключение
      CALL  SOUND
      CALL  SOUND
      JMP   SET_REAL
EX03 ENDP
EX04 PROC NEAR
      MOV   _DL,56  ;4-е исключение
      CALL  SOUND
      CALL  SOUND
      JMP   SET_REAL
EX04 ENDP
EX05 PROC NEAR
      MOV   _DL,57  ;5-е исключение
      CALL  SOUND
      CALL  SOUND
      JMP   SET_REAL
EX05 ENDP
EX06 PROC NEAR
```

```
      MOV   _DL, 58  ;6-е исключение
      CALL  SOUND
      CALL  SOUND
      JMP   SET_REAL
EX06  ENDP
EX07  PROC  NEAR
      MOV   _DL,59  ;7-е исключение
      CALL  SOUND
      CALL  SOUND
      JMP   SET_REAL
EX07  ENDP
EX08  PROC  NEAR
      MOV   _DL,60  ;8-е исключение
      CALL  SOUND
      CALL  SOUND
      JMP   SET_REAL
EX08  ENDP
EX09  PROC  NEAR
      MOV   _DL,61  ;9-е исключение
      CALL  SOUND
      CALL  SOUND
      JMP   SET_REAL
EX09  ENDP
EXOA  PROC  NEAR
      MOV   _DL,62  ;AH-е исключение
      CALL  SOUND
      CALL  SOUND
      JMP   SET_REAL
EXOA  ENDP
EXOB  PROC  NEAR
      MOV   _DL,63  ;BH-е исключение
      CALL  SOUND
      CALL  SOUND
      JMP   SET_REAL
EXOB  ENDP
EXOC  PROC  NEAR
      MOV   _DL,64  ;CH-е исключение
      CALL  SOUND
      CALL  SOUND
      JMP   SET_REAL
EXOC  ENDP
EXOD  PROC  NEAR
      MOV   _DL,65  ;DH-е исключение
      CALL  SOUND
```

```
       CALL  SOUND
       JMP   SET_REAL
EXOD ENDP
EXOE PROC NEAR
       MOV   _DL,66  ;EH-e исключение
       CALL  SOUND
       CALL  SOUND
       JMP   SET_REAL
EXOE ENDP
EXOF PROC NEAR
       MOV   _DL,67  ;FH-e исключение
       CALL  SOUND
       CALL  SOUND
       JMP   SET_REAL
EXOF ENDP
EX10 PROC NEAR
       MOV   _DL,68  ;10H-e исключение
       CALL  SOUND
       CALL  SOUND
       JMP   SET_REAL
EX10 ENDP
EX11 PROC NEAR
       MOV   _DL,69  ;11H-e исключение
       CALL  SOUND
       CALL  SOUND
       JMP   SET_REAL
EX11 ENDP
EX12 PROC NEAR
       MOV   _DL,70  ;12H-e исключение
       CALL  SOUND
       CALL  SOUND
       JMP   SET_REAL
EX12 ENDP
EX13 PROC NEAR
       MOV   _DL,71  ;13H-e исключение
       CALL  SOUND
       CALL  SOUND
       JMP   SET_REAL
EX13 ENDP
EX14 PROC NEAR
       MOV   _DL,72  ;14H-e исключение
       CALL  SOUND
       CALL  SOUND
       JMP   SET_REAL
```

```
EX14   ENDP
EX15   PROC  NEAR
        MOV   _DL,73  ;15H-е  исключение
        CALL  SOUND
        CALL  SOUND
        JMP   SET_REAL
EX15   ENDP
EX16   PROC  NEAR
        MOV   _DL,74  ;16H-е  исключение
        CALL  SOUND
        CALL  SOUND
        JMP   SET_REAL
EX16   ENDP
EX17   PROC  NEAR
        MOV   _DL,75  ;17H-е  исключение
        CALL  SOUND
        CALL  SOUND
        JMP   SET_REAL
EX17   ENDP
EX18   PROC  NEAR
        MOV   _DL,76  ;18H-е  исключение
        CALL  SOUND
        CALL  SOUND
        JMP   SET_REAL
EX18   ENDP
EX19   PROC  NEAR
        MOV   _DL,77  ;19H-е  исключение
        CALL  SOUND
        CALL  SOUND
        JMP   SET_REAL
EX19   ENDP
EX1A   PROC  NEAR
        MOV   _DL,78  ;1AH-е  исключение
        CALL  SOUND
        CALL  SOUND
        JMP   SET_REAL
EX1A   ENDP
EX1B   PROC  NEAR
        MOV   _DL,79  ;1BH-е  исключение
        CALL  SOUND
        CALL  SOUND
        JMP   SET_REAL
EX1B   ENDP
EX1C   PROC  NEAR
```

```
        MOV   _DL,80  ;1CH-e исключение
        CALL  SOUND
        CALL  SOUND
        JMP   SET_REAL
EX1C  ENDP
EX1D  PROC NEAR
        MOV   _DL,81  ;1DH-e исключение
        CALL  SOUND
        CALL  SOUND
        JMP   SET_REAL
EX1D ENDP
EX1E  PROC NEAR
        MOV   _DL,82  ;1EH-e исключение
        CALL  SOUND
        CALL  SOUND
        JMP   SET_REAL
EX1E ENDP
EX1F  PROC NEAR
        MOV   _DL,83  ;1FH-e исключение
        CALL  SOUND
        CALL  SOUND
        JMP   SET_REAL
EX1F ENDP
PAUSE  PROC
        PUSH CX
        MOV   CX,550
PL:
        PUSH  CX
        MOV   CX,0FFFFH
PL1:
        LOOP  PL1
        POP   CX
        LOOP  PL
        POP   CX
        RETN
PAUSE  ENDP
;сообщение об ошибке и возврат в реальный режим
;возврат в реальный режим
SET_REAL  PROC  NEAR
        CLI
        CALL  DISABLE
;запомнить содержимое регистра стека
        MOV  _SP,SP
```

```
;сброс процессора
      MOV AL,0FEH
      OUT 64H,AL
;ожидание возврата по метке SHUT_DOWN_RETURN
_WAIT:
      JMP _WAIT
;--теперь мы в реальном режиме--
;метка возврата
SHUT_DOWN_RETURN:
;восстанавливаем значения всех сегментных регистров
      MOV AX,DATA
      MOV DS,AX
      MOV SS,_SS
      MOV SP,_SP
      MOV ES,_ES
;немаскируемые прерывания
      MOV  AX,0DH
      OUT  70H,AL
;маскируемые прерывания на уровне контроллера прерываний
      XOR  AL,AL
      OUT  21H,AL
      OUT  0A1H,AL
;маскируемые прерывания на уровне микропроцессора
      STI
      MOV  DL,_DL
      MOV  AH,2
      INT  21H
      MOV  AH,4CH
      INT 21H
      RETN
SET_REAL ENDP
;процедура инициализации защищенного режима
INI_PROT PROC
;заполнение таблицы прерываний
;заполнение GDT
;----------------
      MOV AX,DATA
      MOV DS,AX
      MOV DL,AH
      SHR DL,4
      SHL AX,4
      MOV SI,AX
      MOV DI,DX
```

```
;-----------дескриптор для GDT
      ADD AX,OFFSET GDT0
      ADC DL,0
      MOV BX,OFFSET GDT_GDT
      MOV WORD PTR [BX] ,GDT_SIZE-1
      MOV  [BX+2],AX
      MOV  [BX+4],DL
      MOV BYTE PTR  [BX+5],DATA_AC
;-------------дескриптор сегмента данных
      MOV BX,OFFSET GDT_DS
      MOV AX,SI
      MOV DX,DI
      MOV WORD PTR [BX] ,DSEG_SIZE-1
      MOV  [BX+2],AX
      MOV  [BX+4],DL
      MOV BYTE PTR  [BX+5],DATA_AC
;-----------дескриптор сегмента кода
      MOV BX,OFFSET GDT_CS
      MOV AX,CS
      MOV DL,AH
      SHR DL,4
      SHL AX,4
      MOV WORD PTR  [BX],CSEG_SIZE-1
      MOV  [BX+2],AX
      MOV  [BX+4],DL
      MOV BYTE PTR  [BX+5],CODE_AC
;----------------
      MOV BX,OFFSET GDT_SS
      MOV AX,SS
      MOV DL,AH
      SHR DL,4
      SHL AX,4
      MOV WORD PTR  [BX],400-1
      MOV  [BX+2],AX
      MOV  [BX+4],DL
      MOV BYTE PTR  [BX+5],DATA_AC
;----------------
      MOV BX,OFFSET GDT_CRT
      MOV WORD PTR  [BX],3999
      MOV WORD PTR  [BX+2],8000H
      MOV BYTE PTR  [BX+4],0BH
      MOV BYTE PTR  [BX+5],DATA_AC
;----------------
```

```
        MOV BX,OFFSET GDT_MDA
        MOV WORD PTR [BX],3999
        MOV BYTE PTR [BX+4],0BH
        MOV BYTE PTR [BX+5],DATA_AC
;-----------------структура для таблицы прерываний
        MOV BX,OFFSET GDT_IDT
        MOV AX,DS
        MOV DL,AH
        SHR DL,4
        SHL AX,4
        ADD AX,OFFSET EXC_00
        ADC DL,0
        MOV WORD PTR [BX],IDT_SIZE-1
        MOV WORD PTR [BX+2],AX
        MOV BYTE PTR [BX+4],DL
        MOV BYTE PTR [BX+5],IDT_AC
;-------------------исключения
        MOV AX,OFFSET EX00
        MOV BX,OFFSET EXC_00
        MOV [BX],AX
        MOV WORD PTR [BX+2] , (GDT_CS-GDT0)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;-------------------------------
        MOV AX,OFFSET EX01
        MOV BX,OFFSET EXC_01
        MOV [BX],AX
        MOV WORD PTR [BX+2] , (GDT_CS-GDT0)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;-------------------------------
        MOV AX,OFFSET EX02
        MOV BX,OFFSET EXC_02
        MOV [BX],AX
        MOV WORD PTR [BX+2] , (GDT_CS-GDT0)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;-------------------------------
        MOV AX,OFFSET EX03
        MOV BX,OFFSET EXC_03
        MOV [BX],AX
        MOV WORD PTR [BX+2],(GDT_CS-GDT0)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
```

```
;------------------------------
      MOV AX,OFFSET EX04
      MOV BX,OFFSET EXC_04
      MOV [BX],AX
      MOV WORD PTR [BX+2], (GDT_CS-GDTO)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
      MOV AX,OFFSET EX05
      MOV BX,OFFSET EXC_05
      MOV [BX],AX
      MOV WORD PTR [BX+2],(GDT_CS-GDT0)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
      MOV AX,OFFSET EX06
      MOV BX,OFFSET EXC_06
      MOV [BX],AX
      MOV WORD PTR [BX+2],(GDT_CS-GDT0)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
      MOV AX,OFFSET EX07
      MOV BX,OFFSET EXC_07
      MOV [BX],AX
      MOV WORD PTR [BX+2],(GDT_CS-GDT0)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
      MOV AX,OFFSET EX08
      MOV BX,OFFSET EXC_08
      MOV [BX],AX
      MOV WORD PTR [BX+2], (GDT_CS-GDTO)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
      MOV AX,OFFSET EX09
      MOV BX,OFFSET EXC_09
      MOV [BX],AX
      MOV WORD PTR [BX+2], (GDT_CS-GDTO)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
      MOV AX,OFFSET EXOA
      MOV BX,OFFSET EXC_OA
```

```
        MOV  [BX],AX
        MOV WORD PTR [BX+2] , (GDT_CS-GDTO)
        MOV BYTE PTR  [BX+4],0
        MOV BYTE PTR  [BX+5],TRAP_AC
;--------------------------------
        MOV AX,OFFSET EXOB
        MOV BX,OFFSET EXC_OB
        MOV  [BX],AX
        MOV WORD PTR  [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR  [BX+4],0
        MOV BYTE PTR  [BX+5] ,TRAP_AC
;--------------------------------
        MOV AX,OFFSET EXOC
        MOV BX,OFFSET EXC_OC
        MOV  [BX],AX
        MOV WORD PTR  [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR  [BX+4],0
        MOV BYTE PTR  [BX+5],TRAP_AC
;--------------------------------
        MOV AX,OFFSET EXOD
        MOV BX,OFFSET EXC_OD
        MOV  [BX],AX
        MOV WORD PTR  [BX+2],(GDT_CS-GDT0)
        MOV BYTE PTR  [BX+4],0
        MOV BYTE PTR  [BX+5],TRAP_AC
;--------------------------------
        MOV AX,OFFSET EXOE
        MOV BX,OFFSET EXC_OE
        MOV  [BX],AX
        MOV WORD PTR  [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR  [BX+4],0
        MOV BYTE PTR  [BX+5],TRAP_AC
;--------------------------------
        MOV AX,OFFSET EXOF
        MOV BX,OFFSET EXC_OF
        MOV  [BX],AX
        MOV WORD PTR  [BX+2],(GDT_CS-GDT0)
        MOV BYTE PTR  [BX+4],0
        MOV BYTE PTR  [BX+5],TRAP_AC
;--------------------------------
        MOV AX,OFFSET EX10
        MOV BX,OFFSET EXC_10
        MOV  [BX],AX
        MOV WORD PTR  [BX+2], (GDT_CS-GDTO)
```

```
        MOV  BYTE  PTR  [BX+4],0
        MOV  BYTE  PTR  [BX+5],TRAP_AC
;------------------------------
        MOV  AX,OFFSET  EX11
        MOV  BX,OFFSET  EXC_11
        MOV  [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDTO)
        MOV  BYTE  PTR  [BX+4],0
        MOV  BYTE  PTR  [BX+5],TRAP_AC
;------------------------------
        MOV  AX,OFFSET  EX12
        MOV  BX,OFFSET  EXC_12
        MOV   [BX],AX
        MOV  WORD  PTR  [BX+2],(GDT_CS-GDT0)
        MOV  BYTE  PTR  [BX+4],0
        MOV  BYTE  PTR  [BX+5],TRAP_AC
;------------------------------
        MOV  AX,OFFSET  EX13
        MOV  BX,OFFSET  EXC_13
        MOV   [BX],AX
        MOV  WORD  PTR  [BX+2],(GDT_CS-GDT0)
        MOV  BYTE  PTR  [BX+4],0
        MOV  BYTE  PTR  [BX+5],TRAP_AC
;------------------------------
        MOV  AX,OFFSET  EX14
        MOV  BX,OFFSET  EXC_14
        MOV   [BX],AX
        MOV  WORD  PTR  [BX+2], (GDT_CS-GDT0)
        MOV  BYTE  PTR  [BX+4],0
        MOV  BYTE  PTR  [BX+5],TRAP_AC
;------------------------------
        MOV  AX,OFFSET  EX15
        MOV  BX,OFFSET  EXC_15
        MOV   [BX],AX
        MOV  WORD  PTR  [BX+2],(GDT_CS-GDT0)
        MOV  BYTE  PTR  [BX+4],0
        MOV  BYTE  PTR  [BX+5],TRAP_AC
;------------------------------
        MOV  AX,OFFSET  EX16
        MOV  BX,OFFSET  EXC_16
        MOV   [BX],AX
        MOV WORD PTR [BX+2] , (GDT_CS-GDTO)
        MOV  BYTE  PTR  [BX+4],0
        MOV  BYTE  PTR  [BX+5],TRAP_AC
```

```
;-------------------------------
      MOV AX,OFFSET EX17
      MOV BX,OFFSET EXC_17
      MOV  [BX],AX
      MOV WORD PTR [BX+2],(GDT_CS-GDT0)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],TRAP_AC
;-------------------------------
      MOV AX,OFFSET EX18
      MOV BX,OFFSET EXC_18
      MOV  [BX],AX
      MOV WORD PTR [BX+2],(GDT_CS-GDT0)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],TRAP_AC
;-------------------------------
      MOV AX,OFFSET EX19
      MOV BX,OFFSET EXC_19
      MOV  [BX],AX
      MOV WORD PTR [BX+2],(GDT_CS-GDT0)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],TRAP_AC
;-------------------------------
      MOV AX,OFFSET EX1A
      MOV BX,OFFSET EXC_1A
      MOV  [BX],AX
      MOV WORD PTR [BX+2],(GDT_CS-GDT0)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],TRAP_AC
;-------------------------------
      MOV AX,OFFSET EX1B
      MOV BX,OFFSET EXC_1B
      MOV  [BX],AX
      MOV WORD PTR [BX+2],(GDT_CS-GDT0)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],TRAP_AC
;-------------------------------
      MOV AX,OFFSET EX1C
      MOV BX,OFFSET EXC_1C
      MOV  [BX],AX
      MOV WORD PTR [BX+2],(GDT_CS-GDT0)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],TRAP_AC
;-------------------------------
      MOV AX,OFFSET EX1D
      MOV BX,OFFSET EXC_1D
```

```
        MOV  [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR  [BX+4],0
        MOV BYTE PTR  [BX+5],TRAP_AC
;-------------------------------
        MOV AX,OFFSET EX1E
        MOV BX,OFFSET EXC_1E
        MOV  [BX],AX
        MOV WORD PTR  [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR  [BX+4],0
        MOV BYTE PTR [BX+5] ,TRAP_AC
;-------------------------------
        MOV AX,OFFSET EX1F
        MOV BX,OFFSET EXC_1F
        MOV  [BX],AX
        MOV WORD PTR  [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR  [BX+4],0
        MOV BYTE PTR  [BX+5],TRAP_AC
;----———————прерывания
        MOV AX,OFFSET TIMER
        MOV BX,OFFSET INT_20
        MOV  [BX],AX
        MOV WORD PTR  [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR  [BX+4],0
        MOV BYTE PTR  [BX+5],INT_AC
;-------------------------------
        MOV AX,OFFSET KEY
        MOV BX,OFFSET INT_21
        MOV  [BX],AX
        MOV WORD PTR  [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR  [BX+4],0
        MOV BYTE PTR  [BX+5],INT_AC
;-------------------------------
        MOV AX,OFFSET INT1
        MOV BX,OFFSET INT_22
        MOV  [BX],AX
        MOV WORD PTR  [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR  [BX+4],0
        MOV BYTE PTR  [BX+5],INT_AC
;-------------------------------
        MOV AX,OFFSET INT1
        MOV BX,OFFSET INT_23
        MOV  [BX],AX
        MOV WORD PTR  [BX+2], (GDT_CS-GDTO)
```

```
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
;-------------------------------
        MOV AX,OFFSET INT1
        MOV BX,OFFSET INT_24
        MOV [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
;-------------------------------
        MOV AX,OFFSET INT1
        MOV BX,OFFSET INT_25
        MOV [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
;-------------------------------
        MOV AX,OFFSET INT1
        MOV BX,OFFSET INT_26
        MOV [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
;-------------------------------
        MOV AX,OFFSET INT1
        MOV BX,OFFSET INT_27
        MOV [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
;-------------------------------
        MOV AX,OFFSET INT2
        MOV BX,OFFSET INT_28
        MOV [BX],AX
        MOV WORD PTR [BX+2],(GDT_CS-GDT0)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
;-------------------------------
        MOV AX,OFFSET INT2
        MOV BX,OFFSET INT_29
        MOV [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
```

```
;-------------------------------
        MOV AX,OFFSET INT2
        MOV BX,OFFSET INT_2A
        MOV [BX],AX
        MOV WORD PTR [BX+2],(GDT_CS-GDT0)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
;-------------------------------
        MOV AX,OFFSET INT2
        MOV BX,OFFSET INT_2B
        MOV [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
;-------------------------------
        MOV AX,OFFSET INT2
        MOV BX,OFFSET INT_2C
        MOV [BX],AX
        MOV WORD PTR [BX+2],(GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
;-------------------------------
        MOV AX,OFFSET INT2
        MOV BX,OFFSET INT_2D
        MOV [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
;-------------------------------
        MOV AX,OFFSET INT2
        MOV BX,OFFSET INT_2E
        MOV [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDT0)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
;-------------------------------
        MOV AX,OFFSET INT2
        MOV BX,OFFSET INT_2F
        MOV [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
```

```
;———————————адрес возврата из защищенного режима
        PUSH DS
        MOV  AX,40H
        MOV  DS,AX
        MOV  WORD PTR DS:[67H],OFFSET SHUT_DOWN_RETURN
        MOV  DS:[69H],CS
        POP  DS
;записываем в ячейку OFH CMOS код 5
;чтобы обеспечить возврат после сброса процессора
        CLI
        MOV AL,8FH
        OUT 70H,AL
        JMP $+2
        MOV AL,5
        OUT 71H,AL
        RETN
INI_PROT  ENDP
CSEG_SIZE=$-BEGIN
CODE  ENDS
        END BEGIN
```

Рис. 20.3. Пример программы, работающей в защищенном режиме с обработкой прерываний.

Разберем программу на Рис. 20.3. Если говорить о структуре данных программы, то отличие от двух предыдущих программ данной главы заключается в том, что появилась еще таблица прерываний. Заполнение этой таблицы, как и остальных дескрипторов, производится процедурой INI_PROT. Каждому элементу таблицы прерываний соответствует своя процедура обработки. При возникновении исключения производится выход из программы с передачей кода ошибки в ячейку _DL. Особо обрабатываются прерывания от таймера и клавиатуры. Процедура обработки таймера производит звуковой сигнал приблизительно раз в секунду. Звуковой сигнал включается и при нажатии клавиши. При нажатии клавиши ESC программа прерывается и происходит выход в реальный режим. Программа максимально упрощена, поэтому при выходе выводится не код ошибки, а символ, соответствующий данному коду. Чтобы проверить работу исключений, нужно перед меткой _WAI поставить какую-либо команду, вызывающую ошибку. Такой командой может быть, например, команда записи в сегмент кода: MOV WORD PTR CS:_WAIT,90. Выполнить процедуру исключения можно и непосредственно командой INT. Отсюда ясно, как создавать новые прерывания. Достаточно добавить к таблице прерывания еще один дескриптор, указывающий на соответствующую процедуру. Номер вектора определится порядковым номером данного дескриптора в таблице. Вызов такого прерывания, естественно, будет осуществляться командой INT.

И последний пример данной главы. Часть, обрабатывающая прерывания, практически совпадает с аналогичной частью в предыдущей программе. Что же здесь нового? Во-первых, вход и выход из защищенного режима производится командами 386-го

процессора. Во-вторых, здесь демонстрируется, как в принципе можно сочетать защищенный режим и работу с файлами.

Обратимся вначале к командам 386-го процессора. Переход в защищенный режим производится установкой бита 0 в регистре CRO (см. главу 20). Вот и все, все остальное так же, как и для 286-го процессора. Выход из защищенного режима несколько сложнее. Мы действовали согласно алгоритму, приведенному во многих книжках по 386-му и 486-му процессорам (см. [19, 22, 25]). Вот этот алгоритм:

1. Если разрешено страничное преобразование, выполняются следующие операции:
   - передать управление линейным адресам, имеющим тождественное отображение, т.е. линейные адреса равны физическим адресам;
   - сбросить бит PG в регистре CRO;
   - передать ноль в регистр CR3 для очистки буфера страниц.

   В нашем случае страничная адресация не вводилась, поэтому первый пункт можно не выполнять.
2. Передать управление сегменту, который имеет предел 64К. При этом в регистр CS загружается предел сегмента, который он должен иметь в реальном режиме. В нашем случае изначально в регистр CS заложен селектор с указанными параметрами.
3. Загрузить в сегментные регистры SS, DS, ES, FS и GS селектор для для дескриптора, содержащего следующие значения:
   - предел 64К,
   - байтная гранулярность (G=0),
   - расширение вверх,
   - сегмент записываемый,
   - присутствующий в памяти,
   - база - любое значение.

   В нашем случае регистры SS, DS, ES изначально загружаются такими селекторами. Регистры же FS, GS мы нигде не используем.
4. Запретить прерывания.
5. Сбросить нулевой бит в CRO.
6. Перейти к программе реального режима путем дальнего перехода, для очистки очереди.
7. Загрузить регистр прерываний. Перед выходом в защищенный режим мы сохранили этот регистр, а потом восстановили. Кроме того, нужно снова перепрограммировать контроллер прерываний для реального режима.
8. Разрешить прерывания.
9. Загрузить сегментные регистры.

Именно по такой схеме мы действовали в данном случае. Читатель уже, наверное, задал себе важный вопрос, как работать с файлами в защищенном режиме. Автор видит три пути решения данной проблемы:

1. Каким-то образом, находясь в защищенном режиме, использовать стандартные процедуры DOS. Наверное, в силу своей близорукости я не вижу, как это сделать достаточно просто.

2. Написать собственную библиотеку работы с файлами или использовать стандартныебиблиотеки.
3. Перед обращением к какой-либо DOS-овской процедуре выйти в реальный режим, а потом снова вернуться в защищенный.

Последний подход и реализован в данном примере. Может возникнуть вопрос, как в защищенном режиме загружать большие объемы информации. Это можно сделать через промежуточный буфер, который будет находиться в сегменте данных, доступ к которому, естественно, будет как в защищенном, так и в реальном режиме. Похоже, что именно такой подход был реализован и в старой Windows 3.1.

```
;работаем в защищенном режиме 386-го процессора
.386P
;байты доступа
;10000000B - сегмент есть в памяти
;00011000B - сегмент кода
;00010000B - сегмент данных
;00000100B - сегмент расширяется вниз
;00000100B - согласованный сегмент
;00000010B - разрешена запись
;00000110B - вентиль прерывания
;00000111B - вентиль исключения
;сегмент данных
DATA_AC    EQU 10010010B
;сегмент кода
CODE_AC    EQU 10011000B
;сегмент стека
STACK_AC   EQU 10010110B
;сегмент таблицы IDT
IDT_AC     EQU 10010010B
;байт доступа вентиля прерывания
INT_AC     EQU 10000110B
;байт доступа вентиля исключения
TRAP_AC    EQU 10000111B
;сегмент стека
ST1 SEGMENT STACK 'STACK'
      DB 100 DUP(?)
ST1 ENDS
;сегмент данных
DATA SEGMENT
BEG_DATA=$
_SS  DW ?
_ES  DW ?
IDTT DQ ?
MSG  DB 'Перешли в защищенный режим.', 0
```

```
MSG1  DB  'Открыли файл и перешли в защищенный режим.  ',0
MSG2  DB  'Записали в файл и перешли в защищенный режим.  ',0
MSG3  DB  ' Закрыли файл и перешли в защищенный режим.  ',0
PATH  DB  'PROT.TXT',0
BUF   DB  'Файл проверки'
_DL DB 0
/здесь будет храниться дескриптор файла
_AX DW ?
/данные для вывода на экран
COLUMNS  DB ?          ;количество столбцов на экране
ROWS     DB ?          /количество строк на экране
SEL_BUF  DW ?          ;селектор видеобуфера
SEG_BUF  DW ?          /сегментный адрес видеобуфера
;глобальная дескрипторная таблица
/нулевой дескриптор
GDT0      DQ O
;дескриптор для GDT
GDT_GDT   DQ O
/дескриптор для сегмента кода
GDT_CS    DQ O
/дескриптор для сегмента данных
GDT_DS    DQ O
/дескриптор для сегмента стека
GDT_SS    DQ O
;дескриптор для видеопамяти цветного дисплея
GDT_CRT   DQ O
/дескриптор для видеопамяти монохромного дисплея
GDT_MDA   DQ O
;дескриптор таблицы прерываний
GDT_IDT   DQ 0
GDT_SIZE = $-GDTO   /размер глобальной дескрипторной таблицы
;вентили исключений
IDT_BEG=$
EXC_00     DQ  0
EXC_01     DQ  0
EXC_02     DQ  0
EXC_03     DQ  0
EXC_04     DQ  0
EXC_05     DQ  0
EXC_06     DQ  0
EXC_07     DQ  0
EXC_08     DQ  0
EXC_09     DQ  0
EXC_OA     DQ  0
EXC_OB     DQ  0
```

```
EXC_0C     DQ 0
EXC_0D     DQ 0
EXC_0E     DQ 0
EXC_0F     DQ 0
EXC_10     DQ 0
EXC_11     DQ 0
EXC_12     DQ 0
EXC_13     DQ 0
EXC_14     DQ 0
EXC_15     DQ 0
EXC_16     DQ 0
EXC_17     DQ 0
EXC_18     DQ 0
EXC_19     DQ 0
EXC_1A     DQ 0
EXC_1B     DQ 0
EXC_1C     DQ 0
EXC_1D     DQ 0
EXC_1E     DQ 0
EXC_1F     DQ 0
;вентили прерываний
INT_20     DQ 0 ;таймер
INT_21     DQ 0 ;клавиатура
INT_22     DQ 0
INT_23     DQ 0
INT_24     DQ 0
INT_25     DQ 0
INT_26     DQ 0
INT_27     DQ 0
INT_28     DQ 0
INT_29     DQ 0
INT_2A     DQ 0
INT_2B     DQ 0
INT_2C     DQ 0
INT_2D     DQ 0
INT_2E     DQ 0
INT_2F     DQ 0
IDT_SIZE=$-IDT_BEG   ;размер таблицы прерываний
;-------------------------------
DSEG_SIZE=$-BEG_DATA ;размер сегмента данных
DATA ENDS
;сегмент кода
CODE SEGMENT PUBLIC PARA USE16
      ASSUME CS:CODE, DS:DATA, SS:ST1
```

```
BEGIN:
      MOV AX,DATA
      MOV DS,AX
      MOV _DL,0
; сохраняем сегменты
      MOV _ES,ES
      MOV _SS,SS
      SIDT QWORD PTR IDTT
;------------------------------
      CALL WHAT_CRT
;войти в защищенный режим
      CALL SET_PROT
;очистить экран
      MOV  AH,07H
      CALL CLS
;координаты
      MOV BX, 0  ;координата X
      MOV AX,2   ;координата Y
;адрес выводимой строки
      MOV SI,OFFSET MSG
      MOV AH,40H  ;атрибут
      CALL WRITE
;здесь работаем с файлами
;открыть файл
      CALL  SET_REAL
      MOV   AH,3CH
      LEA   DX,PATH
      MOV   CX,0
      INT   21H
      MOV   _AX,AX
      CALL  SET_PROT
;координаты
      MOV BX,0   /координата X
      MOV AX,3   ;координата Y
;адрес выводимой строки
      MOV SI,OFFSET MSG1
      MOV AH,41H  ;атрибут
      CALL WRITE
/следующая команда, если снять с нее комментарий
/приведет к аварийному выходу из программы
;      MOV BYTE PTR CS:_END,AL
;записать туда
      CALL  SET_REAL
      MOV  BX,_AX
```

```
        LEA   DX,BUF
        MOV   CX,13
        MOV   AH,40H
        INT   21H
        CALL  SET_PROT
;координаты
        MOV BX,0     ;координата X
        MOV AX,4     ;координата Y
;адрес выводимой строки
        MOV SI,OFFSET MSG2
        MOV AH,41H    ;атрибут
        CALL WRITE
;закрыть файл
        CALL  SET_REAL
        MOV   BX,_AX
        MOV   AH,3EH
        INT   21H
        CALL  SET_PROT
;координаты
        MOV BX,0     ;координата X
        MOV AX,5     ;координата Y
;адрес выводимой строки
        MOV SI,OFFSET MSG3
        MOV AH,41H    ;атрибут
        CALL WRITE
;выход
        CALL  SET_REAL
        MOV  _DL,48
_END:
        MOV   DL,_DL
        MOV  AH, 2
        INT   21H
        MOV   AH,4CH
        INT  21H
;раздел процедур
;возврат в реальный режим
SET_REAL  PROC  NEAR
        CLI
        MOV AX,DS
        MOV ES,AX
        MOV   AX,CR0
        AND   AX,0FFFEH
;переход в реальный режим
        MOV   CR0,AX
```

```
;--теперь мы в реальном режиме--
;———JMP FAR FLUSH, для сбрасывания очереди команд
        DB   0EAH
        DW   OFFSET FLUSH
        DW   SEG FLUSH
FLUSH:
;перепрограммировать контроллер прерываний
        CALL PROG_INT1
;восстановить сегментные регистры
        MOV AX,DATA
        MOV DS,AX
        MOV SS,_SS
        MOV ES,_ES
;восстановить регистр прерываний
        LIDT QWORD PTR IDTT
/запретить линию A20
        CALL DISABLE
/разрешить немаскируемое прерывание
        MOV  AX,0DH
        OUT  70H,AL
;---------------------
        STI
        CMP  _DL,0
        JNZ  _END
        RETN
SET_REAL ENDP
/установка защищенного режима
SET_PROT PROC NEAR
;инициализировать дескрипторные таблицы
        CALL INI_PROT
        CLI
/разрешить линию A20
        CALL ENABLE
;перепрограммировать контроллер прерываний
        CALL PROG_INT
;загрузить регистр прерываний
        LIDT QWORD PTR GDT_IDT
;загрузить регистр глобальной дескрипторной таблицы
        LGDT QWORD PTR GDT_GDT
;устанавливаем защищенный режим
        MOV AX,CR0
        OR  AX,1
        MOV CR0,AX
;———JMP FAR FLUSH1, для сбрасывания очереди команд
```

```
        DB    OEAH
        DW    OFFSET FLUSH1
        DW     (GDT_CS-GDT0)
FLUSH1:
;загрузка селекторов
        MOV   AX,(GDT_DS-GDT0)
        MOV   DS,AX
        MOV  AX,SEL_BUF
        MOV  ES,AX
        MOV   AX,(GDT_SS-GDT0)
        MOV   SS,AX
;размаскировать прерывания на уровне контроллера
        XOR   AL,AL
        OUT  21H,AL
        OUT OA1H ,AX
;размаскировать прерывания на уровне микропроцессора
        STI
        RETN
SET_PROT ENDP
;разрешение линии A20
ENABLE PROC NEAR
        MOV   AL,0D1H    ;команда управления линией A20
        OUT   64H,AL     ;порт состояния клавиатуры
        MOV   AL,0DFH    ;открыть A20
        OUT   60H,AL     ;клавиатурный порт
        RETN
ENABLE ENDP
;запрещение линии A20
DISABLE PROC NEAR
        MOV AL,OD1H
        OUT  64H,AL
        MOV AL,0DDH      ;закрыть A20
        OUT  60H,AL
        RETN
DISABLE ENDP
;перепрограммирование контроллера прерываний
PROG_INT PROC NEAR
;первый контроллер
;прерывания 0-7 получают номера
;20H-27H
        MOV AH,20H
        MOV AL,11H
        OUT 20H,AL
        JMP SHORT $+2
```

```
        MOV AL,AH
        OUT 21H,AL
        JMP SHORT $+2
        MOV AL,4
        OUT 21H,AL
        JMP SHORT $+2
        MOV AL,1
        OUT 21H,AL
        JMP SHORT $+2
        MOV AL,0FFH
        OUT 21H,AL
        JMP SHORT $+2
;второй контроллер
;прерывания 8-15 получают номера
;28H-2FH
        MOV AH,28H
        MOV AL,11H
        OUT 0A0H,AL
        JMP SHORT $+2
        MOV AL,AH
        OUT OA1H,AL
        JMP SHORT $+2
        MOV AL,2
        OUT 0A1H,AL
        JMP SHORT $+2
        MOV AL,1
        OUT OA1H,AL
        JMP SHORT $+2
        MOV AL,0FFH
        OUT 0A1H,AL
        JMP SHORT $+2
        RETN
PROG_INT ENDP
;перепрограммирование контроллера прерываний
PROG_INT1 PROC NEAR
;первый контроллер
;прерывания 0-7 получают номера
        MOV AL,11H
        OUT 20H,AL
        JMP SHORT $+2
        MOV AL, 8
        OUT 21H,AL
        JMP SHORT $+2
        MOV AL,4
```

```
        OUT 21H,AL
        JMP SHORT $+2
        MOV AL,1
        OUT 21H,AL
        JMP SHORT $+2
;второй контроллер
;прерывания 8-15 получают номера
        MOV AL,11H
        OUT 0A0H,AL
        JMP SHORT $+2
        MOV AL,70H
        OUT 0A1H,AL
        JMP SHORT $+2
        MOV AL,2
        OUT 0A1H,AL
        JMP SHORT $+2
        MOV AL,1
        OUT 0A1H,AL
        JMP SHORT $+2
        RETN
PROG_INT1 ENDP
;определение параметров видеосистемы
WHAT_CRT PROC NEAR
        MOV AX,40H
        MOV ES,AX
        MOV BX,ES:[4AH]
        MOV COLUMNS,BL
        MOV BL,ES:[84H]
        INC BL
        MOV ROWS,BL
        MOV BX,ES:[63H]
        CMP BX,3D4H
        JNE NO_COLOR
;цветной адаптер
        MOV SEL_BUF,(GDT_CRT-GDT0)
        MOV SEG_BUF,0B800H
        MOV ES,SEG_BUF
        JMP SHORT EXIT
NO_COLOR:
;адаптер MDA
        MOV SEL_BUF,(GDT_MDA-GDT0)
        MOV SEG_BUF,0B000H
        MOV ES,SEG_BUF
EXIT:
        RETN
```

```
WHAT_CRT ENDP
;вывод на экран строки
;в ES либо селектор экрана, либо сегмент
;AX,BX - X,Y
;SI - адрес строки
;AH - атрибут
WRITE PROC NEAR
;расчет смещения в видеопамяти
        PUSH AX
        MOV  DL,COLUMNS
        MUL  DL
        ADD  AX,BX
        SHL  AX,1
        MOV  DI,AX
        POP  AX
L01:
        LODSB
        CMP  AL,0
        JZ   EN
        STOSW
        JMP  SHORT L01
EN:
        RETN
WRITE ENDP
;обработка клавиатурного прерывания
END_INT PROC NEAR
        IN   AL,61H
        MOV  AH,AL
        OR   AL,80H
        OUT  61H,AL
        XCHG AH,AL
        OUT  61H,AL
        MOV  AL,20H
        OUT  20H,AL
        RETN
END_INT ENDP
KEY PROC NEAR
        PUSH AX
;-------------конец прерывания
        CALL END_INT
        POP  AX
        IRET
KEY ENDP
```

```
;обработка прерывания таймера
;вызывает звуковой сигнал, примерно раз в секунду
TIMER PROC NEAR
      CLI
      MOV  AL,20H
      OUT  20H,AL
      IRET
TIMER ENDP
;обработчик прерываний от первого контроллера
INT1 PROC NEAR
      PUSH AX
      MOV  AL,20H
      OUT  20H,AL
      POP  AX
      MOV  _DL,50
      IRET
INT1 ENDP
;обработчик прерываний от второго контроллера
INT2 PROC NEAR
      PUSH AX
      MOV  AL,20H
      OUT  20H,AL
      OUT  0AH,AL
      POP  AX
      MOV  _DL,51
      IRET
INT2 ENDP
;обработка исключений
EX00 PROC NEAR
      MOV  _DL,52 ;0-е исключение
      JMP  SET_REAL
EX00 ENDP
EX01 PROC NEAR
      MOV  _DL,53 ;1-е исключение
      JMP  SET_REAL
EX01 ENDP
EX02 PROC NEAR
      MOV  _DL,54 ;2-е исключение
      JMP  SET_REAL
EX02 ENDP
EX03 PROC NEAR
      MOV  _DL,55 ;3-е исключение
      JMP  SET_REAL
EX03 ENDP
```

```
EX04  PROC NEAR
       MOV   _DL, 56 ;4-е исключение
       JMP   SET_REAL
EX04  ENDP
EX05  PROC NEAR
       MOV   _DL, 57 ;5-е исключение
       JMP   SET_REAL
EX05  ENDP
EX06  PROC NEAR
       MOV   _DL, 58 ;6-е исключение
       JMP   SET_REAL
EX06  ENDP
EX07  PROC NEAR
       MOV   _DL, 59 ;7-е исключение
       JMP   SET_REAL
EX07  ENDP
EX08  PROC NEAR
       MOV   _DL, 60 ;8-е исключение
       JMP   SET_REAL
EX08  ENDP
EX09  PROC NEAR
       MOV   _DL, 61 ;9-е исключение
       JMP   SET_REAL
EX09  ENDP
EXOA  PROC NEAR
       MOV   _DL, 62 ;AH-е  исключение
       JMP   SET_REAL
EXOA  ENDP
EXOB  PROC NEAR
       MOV   _DL, 63 ;BH-е  исключение
       JMP   SET_REAL
EXOB  ENDP
EXOC  PROC NEAR
       MOV   _DL, 64 ;CH-е  исключение
       JMP   SET_REAL
EXOC  ENDP
EXOD  PROC NEAR
       MOV   _DL, 65 ;DH-е  исключение
       JMP   SET_REAL
EXOD  ENDP
EXOE  PROC NEAR
       MOV   _DL, 66 ;EH-е  исключение
       JMP   SET_REAL
EXOE  ENDP
```

```
EXOF PROC NEAR
      MOV  _DL,67  ;FH-e исключение
      JMP  SET_REAL
EXOF ENDP
EX10 PROC NEAR
      MOV  _DL,68  ;10H-e исключение
      JMP  SET_REAL
EX10 ENDP
EX11 PROC NEAR
      MOV  _DL,69  ;11H-e исключение
      JMP  SET_REAL
EX11 ENDP
EX12 PROC NEAR
      MOV  _DL,70  ;12H-e исключение
      JMP  SET_REAL
EX12 ENDP
EX13 PROC NEAR
      MOV  _DL,71  ;13H-e исключение
      JMP  SET_REAL
EX13 ENDP
EX14 PROC NEAR
      MOV  _DL,72  ;14H-e исключение
      JMP  SET_REAL
EX14 ENDP
EX15 PROC NEAR
      MOV  _DL,73  ;15H-e исключение
      JMP  SET_REAL
EX15 ENDP
EX16 PROC NEAR
      MOV  _DL,74  ;16H-e исключение
      JMP  SET_REAL
EX16 ENDP
EX17 PROC NEAR
      MOV  _DL,75  ;17H-e исключение
      JMP  SET_REAL
EX17 ENDP
EX18 PROC NEAR
      MOV  _DL,76  ;18H-e исключение
      JMP  SET_REAL
EX18 ENDP
EX19 PROC NEAR
      MOV  _DL,77  ;19H-e исключение
      JMP  SET_REAL
EX19 ENDP
```

```
EX1A PROC NEAR
      MOV   _DL,78  ;1AH-е исключение
      JMP   SET_REAL
EX1A ENDP
EX1B PROC NEAR
      MOV   _DL,79  ;1BH-е исключение
      JMP   SET_REAL
EX1B ENDP
EX1C PROC NEAR
      MOV   _DL,80  ;1CH-е исключение
      JMP   SET_REAL
EX1C ENDP
EX1D PROC NEAR
      MOV   _DL,81  ;1DH-е исключение
      JMP   SET_REAL
EX1D ENDP
EX1E PROC NEAR
      MOV   _DL,82  ;1EH-е исключение
      JMP   SET_REAL
EX1E ENDP
EX1F PROC NEAR
      MOV   _DL,83  ;1FH-е исключение
      JMP   SET_REAL
EX1F ENDP
PAUSE  PROC
      PUSH CX
      MOV  CX,550
PL:
      PUSH CX
      MOV   CX,OFFFFH
PL1:
      LOOP PL1
      POP  CX
      LOOP PL
      POP  CX
      RETN
PAUSE  ENDP
;сообщение об ошибке и возврат в реальный режим
;процедура инициализации защищенного режима
INI_PROT PROC
;заполнение таблицы прерываний
;заполнение GDT
;---------------
```

```
        MOV AX,DATA
        MOV DS,AX
        MOV DL,AH
        SHR DL,4
        SHL AX,4
        MOV SI,AX
        MOV DI,DX
;-----------------дескриптор для GDT
        ADD AX,OFFSET GDT0
        ADC DL,0
        MOV BX,OFFSET GDT_GDT
        MOV WORD PTR [BX],0FFFFH
        MOV  [BX+2],AX
        MOV  [BX+4],DL
        MOV BYTE PTR [BX+5],DATA_AC
;-----------------дескриптор сегмента данных
        MOV BX,OFFSET GDT_DS
        MOV AX,SI
        MOV DX,DI
        MOV WORD PTR [BX],0FFFFH
        MOV  [BX+2],AX
        MOV  [BX+4],DL
        MOV BYTE PTR [BX+5],DATA_AC
;--------------дескриптор сегмента кода
        MOV BX,OFFSET GDT_CS
        MOV AX,CS
        MOV DL,AH
        SHR DL,4
        SHL AX,4
        MOV  WORD PTR [BX],0FFFFH
        MOV  [BX+2],AX
        MOV  [BX+4],DL
        MOV BYTE PTR [BX+5],CODE_AC
;----------------
        MOV BX,OFFSET GDT_SS
        MOV AX,SS
        MOV DL,AH
        SHR DL,4
        SHL AX,4
        MOV  WORD PTR [BX],0FFFFH
        MOV  [BX+2],AX
        MOV  [BX+4],DL
        MOV BYTE PTR [BX+5],DATA_AC
;----------------
```

```
        MOV BX,OFFSET GDT_CRT
        MOV WORD PTR [BX],3999
        MOV WORD PTR [BX+2],8000H
        MOV BYTE PTR [BX+4],0BH
        MOV BYTE PTR [BX+5],DATA_AC
;----------------
        MOV BX,OFFSET GDT_MDA
        MOV WORD PTR [BX],3999
        MOV BYTE PTR [BX+4],0BH
        MOV BYTE PTR [BX+5],DATA_AC
;-----------------структура для таблицы прерываний
        MOV BX,OFFSET GDT_IDT
        MOV AX,DS
        MOV DL,AH
        SHR DL,4
        SHL AX,4
        ADD AX,OFFSET EXC_00
        ADC DL,0
        MOV WORD PTR [BX],IDT_SIZE-1
        MOV WORD PTR [BX+2],AX
        MOV BYTE PTR [BX+4],DL
        MOV BYTE PTR [BX+5],IDT_AC
;-----------------исключения
        MOV AX,OFFSET EXOO
        MOV BX,OFFSET EXC_00
        MOV [BX],AX
        MOV WORD PTR [BX+2],(GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
        MOV AX,OFFSET EX01
        MOV BX,OFFSET EXC_01
        MOV [BX],AX
        MOV WORD PTR [BX+2],(GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
        MOV AX,OFFSET EX02
        MOV BX,OFFSET EXC_02
        MOV [BX],AX
        MOV WORD PTR [BX+2],(GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
```

```
        MOV AX,OFFSET EX03
        MOV BX,OFFSET EXC_03
        MOV [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
        MOV AX,OFFSET EX04
        MOV BX,OFFSET EXC_04
        MOV [BX],AX
        MOV WORD PTR [BX+2] , (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
        MOV AX,OFFSET EX05
        MOV BX,OFFSET EXC_05
        MOV [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
        MOV AX,OFFSET EX06
        MOV BX,OFFSET EXC_06
        MOV [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
        MOV AX,OFFSET EX07
        MOV BX,OFFSET EXC_07
        MOV [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
        MOV AX,OFFSET EX08
        MOV BX,OFFSET EXC_08
        MOV [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
        MOV AX,OFFSET EX09
        MOV BX,OFFSET EXC_09
```

```
        MOV  [BX],AX
        MOV WORD PTR [BX+2],(GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
        MOV AX,OFFSET EXOA
        MOV BX,OFFSET EXC_OA
        MOV  [BX],AX
        MOV WORD PTR [BX+2] , (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
        MOV AX,OFFSET EXOB
        MOV BX,OFFSET EXC_0B
        MOV  [BX],AX
        MOV WORD PTR [BX+2],(GDT_CS-GDT0)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
        MOV AX,OFFSET EXOC
        MOV BX,OFFSET EXC_OC
        MOV  [BX],AX
        MOV WORD PTR [BX+2],(GDT_CS-GDT0)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
        MOV AX,OFFSET EXOD
        MOV BX,OFFSET EXC_0D
        MOV  [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
        MOV AX,OFFSET EXOE
        MOV BX,OFFSET EXC_OE
        MOV  [BX],AX
        MOV WORD PTR [BX+2] , (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
        MOV AX,OFFSET EX0F
        MOV BX,OFFSET EXC_OF
        MOV  [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDTO)
```

```
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;-------------------------------
        MOV AX,OFFSET EX10
        MOV BX,OFFSET EXC_10
        MOV [BX],AX
        MOV WORD PTR [BX+2] , (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;-------------------------------
        MOV AX,OFFSET EX11
        MOV BX,OFFSET EXC_11
       .MOV [BX],AX
        MOV WORD PTR [BX+2] , (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;-------------------------------
        MOV AX,OFFSET EX12
        MOV BX,OFFSET EXC_12
        MOV [BX],AX
        MOV WORD PTR [BX+2] , (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;-------------------------------
        MOV AX,OFFSET EX13
        MOV BX,OFFSET EXC_13
        MOV [BX],AX
        MOV WORD PTR [BX+2],(GDT_CS-GDT0)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;-------------------------------
        MOV AX,OFFSET EX14
        MOV BX,OFFSET EXC_14
        MOV [BX],AX
        MOV WORD PTR [BX+2] , (GDT_CS-GDT0)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
;-------------------------------
        MOV AX,OFFSET EX15
        MOV BX,OFFSET EXC_15
        MOV [BX],AX
        MOV WORD PTR [BX+2],(GDT_CS-GDT0)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],TRAP_AC
```

```
;------------------------------
      MOV AX,OFFSET EX16
      MOV BX,OFFSET EXC_16
      MOV [BX],AX
      MOV WORD PTR [BX+2],(GDT_CS-GDT0)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
      MOV AX,OFFSET EX17
      MOV BX,OFFSET EXC_17
      MOV [BX],AX
      MOV WORD PTR [BX+2],(GDT_CS-GDT0)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
      MOV AX,OFFSET EX18
      MOV BX,OFFSET EXC_18
      MOV [BX],AX
      MOV WORD PTR [BX+2], (GDT_CS-GDTO)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
      MOV AX,OFFSET EX19
      MOV BX,OFFSET EXC_19
      MOV [BX],AX
      MOV WORD PTR [BX+2], (GDT_CS-GDTO)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
      MOV AX,OFFSET EX1A
      MOV BX,OFFSET EXC_1A
      MOV [BX],AX
      MOV WORD PTR [BX+2], (GDT_CS-GDTO)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
      MOV AX,OFFSET EX1B
      MOV BX,OFFSET EXC_1B
      MOV [BX],AX
      MOV WORD PTR [BX+2], (GDT_CS-GDT0)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],TRAP_AC
;------------------------------
      MOV AX,OFFSET EX1C
      MOV BX,OFFSET EXC_1C
```

```
        MOV  [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDT0)
        MOV BYTE PTR  [BX+4],0
        MOV BYTE PTR  [BX+5],TRAP_AC
;------------------------------
        MOV AX,OFFSET EX1D
        MOV BX,OFFSET EXC_1D
        MOV  [BX],AX
        MOV WORD PTR  [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR  [BX+4],0
        MOV BYTE PTR  [BX+5],TRAP_AC
;------------------------------
        MOV AX,OFFSET EX1E
        MOV BX,OFFSET EXC_1E
        MOV  [BX],AX
        MOV WORD PTR  [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR  [BX+4],0
        MOV BYTE PTR  [BX+5],TRAP_AC
;------------------------------
        MOV AX,OFFSET EX1F
        MOV BX,OFFSET EXC_1F
        MOV  [BX],AX
        MOV WORD PTR  [BX+2], (GDT_CS-GDT0)
        MOV BYTE PTR  [BX+4],0
        MOV BYTE PTR  [BX+5],TRAP_AC
;-----------прерывания
        MOV AX,OFFSET TIMER
        MOV BX,OFFSET INT_20
        MOV  [BX],AX
        MOV WORD PTR  [BX+2] , (GDT_CS-GDTO)
        MOV BYTE PTR  [BX+4],0
        MOV BYTE PTR  [BX+5],INT_AC
;------------------------------
        MOV AX,OFFSET KEY
        MOV BX,OFFSET INT_21
        MOV  [BX],AX
        MOV WORD PTR  [BX+2] , (GDT_CS-GDTO)
        MOV BYTE PTR  [BX+4],0
        MOV BYTE PTR  [BX+5],INT_AC
;------------------------------
        MOV AX,OFFSET INT1
        MOV BX,OFFSET INT_22
        MOV  [BX],AX
        MOV WORD PTR  [BX+2], (GDT_CS-GDT0)
```

```
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
;------------------------------
        MOV AX,OFFSET INT1
        MOV BX,OFFSET INT_23
        MOV [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
;------------------------------
        MOV AX,OFFSET INT1
        MOV BX,OFFSET INT_24
        MOV [BX],AX
        MOV WORD PTR [BX+2], (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
;------------------------------
        MOV AX,OFFSET INT1
        MOV BX,OFFSET INT_25
        MOV [BX],AX
        MOV WORD PTR [BX+2] , (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
;------------------------------
        MOV AX,OFFSET INT1
        MOV BX,OFFSET INT_26
        MOV [BX],AX
        MOV WORD PTR [BX+2] , (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
;------------------------------
        MOV AX,OFFSET INT1
        MOV BX,OFFSET INT_27
        MOV [BX],AX
        MOV WORD PTR [BX+2] , (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
;------------------------------
        MOV AX,OFFSET INT2
        MOV BX,OFFSET INT_28
        MOV [BX],AX
        MOV WORD PTR [BX+2] , (GDT_CS-GDTO)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5],INT_AC
```

```
;-----------------------------
      MOV AX,OFFSET INT2
      MOV BX,OFFSET INT_29
      MOV [BX],AX
      MOV WORD PTR [BX+2],(GDT_CS-GDTO)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],INT_AC
;-----------------------------
      MOV AX,OFFSET INT2
      MOV BX,OFFSET INT_2A
      MOV [BX],AX
      MOV WORD PTR [BX+2] , (GDT_CS-GDTO)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],INT_AC
;-----------------------------
      MOV AX,OFFSET INT2
      MOV BX,OFFSET INT_2B
      MOV [BX],AX
      MOV WORD PTR [BX+2], (GDT_CS-GDTO)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],INT_AC
;-----------------------------
      MOV AX,OFFSET INT2
      MOV BX,OFFSET INT_2C
      MOV [BX],AX
      MOV WORD PTR [BX+2],(GDT_CS-GDT0)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],INT_AC
;-----------------------------
      MOV AX,OFFSET INT2
      MOV BX,OFFSET INT_2D
      MOV [BX],AX
      MOV WORD PTR [BX+2], (GDT_CS-GDTO)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],INT_AC
;-----------------------------
      MOV AX,OFFSET INT2
      MOV BX,OFFSET INT_2E
      MOV [BX],AX
      MOV WORD PTR [BX+2] , (GDT_CS-GDTO)
      MOV BYTE PTR [BX+4],0
      MOV BYTE PTR [BX+5],INT_AC
;-----------------------------
```

```
        MOV AX,OFFSET INT2
        MOV BX,OFFSET INT_2F
        MOV [BX],AX
        MOV WORD PTR [BX+2],(GDT_CS-GDT0)
        MOV BYTE PTR [BX+4],0
        MOV BYTE PTR [BX+5] , INT_AC
        RETN
INI_PROT ENDP
;процедура очистки экрана
;в ES находится либо селектор, либо сегмент
;в AH - атрибут
CLS   PROC
        MOV  CX,2000
        MOV  AL,32
        XOR  DI,DI
L2:
        STOSW
        LOOP L2
        RETN
CLS   ENDP
CSEG_SIZE=$-BEGIN
CODE ENDS
        END BEGIN
```

Рис. 20.4. Программа, реализующая вход в защищенный режим и выход из него средствами 386-го процессора.

Глава 27. Программирование VGA- и SVGA-адаптеров.

Сделай же, Боже, так, чтобы все потомство его не имело на земле счастья!

Н.В. Гоголь. Страшная месть.

При написании главы 10 автор ориентировался на устаревший графический режим 640*350*16. Можно назвать эту главу вводной. В данной главе возможности VGA рассматриваются более подробно. При работе над ней я воспользовался некоторыми материалами, предоставленными мне А. Кудрявцевым, хорошим программистом и большим знатоком компьютерной графики. К сожалению, обширная справочная литература по видеоадаптерам изобилует ошибками. Благодаря данным, предоставленным Кудрявцевым, мне в значительной степени удалось избежать огромной работы по проверке справочной информации. В главе имеется материал, и по адаптерам SVGA, что является в настоящее время весьма актуальным.

Я позволю себе начать изложение со справочного материала т.к. основы темы были уже изложены в главе 10.

I. Краткий справочник по адаптерам VGA.

В понятие видеоадаптер входит несколько устройств.

Графические устройства.

1. Графический контроллер осуществляет операции обмена информации между микропроцессором и видеопамятью.
2. Последовательный преобразователь: запоминает данные, читаемые из видеопамяти в течение цикла регенерации, преобразует в последовательный поток бит и затем передает контроллеру атрибутов.
3. Контроллер атрибутов: управляет цветами. Цвет символа или пиксела передается (после некоторых преобразований) на дисплей в соответствие с таблицей цветов.
4. Контроллер ЭЛТ. Вырабатывает сигналы управления работой ЭЛТ (развертки и гашения), определяет формат экрана и размер символа, определяет форму курсора, управляет вертикальной сверткой.
5. Синхронизатор: управляет временными параметрами видеоадаптера.
6. Видеопамять. Видеопамять содержит данные, отображаемые на экране. Стандартный VGA-адаптер содержит объем памяти 256 килобайт. Адаптеры Super VGA могут содержать памяти до 1 мегабайта и выше.

Принцип работы растрового дисплея.

Изображение на экране дисплея формируется посредством группы горизонтальных строк, называемых растром. Растр обновляется электронным лучом приблизительно 50-70 раз в секунду. Такая частота совершенно незаметна для глаз человека, поэтому мы

видим на экране стационарное изображение. Каждая точка (пиксель) цветного экрана состоит из трех цветных люминесцентных точек - красной, зеленой, синей (RGB - Red Green Blue). Электронный луч, в свою очередь, состоит из трех компонент, отвечающих за свой цвет. В зависимости от интенсивности компонент луча каждая точка получает свой результирующий оттенок. Поскольку глаз человека не способен различить отдельные компоненты пикселя, для него точка имеет один определенный цвет.

Луч начинает свое движение из левого верхнего угла экрана. Двигаясь по строке экрана слева направо (прямой горизонтальный ход луча), луч заставляет светиться каждую точку в соответствии с информацией, получаемой от видеоадаптера, которая формируется путем соответствующего кодирования информации, считанной адаптером из видеобуфера. Когда луч доходит до правого края экрана, то происходит его гашение, и он переводится в левый угол следующей строки (обратный горизонтальный ход луча), где снова начинает свое движение в правый угол. Кроме основного изображения, формируется еще горизонтальное окаймление. Правое окаймление связано с тем, что существует небольшой интервал между окончанием прямого горизонтального хода луча и началом обратного горизонтального хода луча. Аналогично левое окаймление связано с существованием малого временного интервала между концом обратного горизонтального хода луча и началом прямого горизонтального хода луча.

После того как луч достигает нижней строки экрана, он гасится на некоторое время, чтобы перевести его снова в левый верхний угол. Такое движение называется вертикальным обратным ходом луча. Затем цикл формирования растра повторяется (цикл регенерации). Существование обратного вертикального хода луча приводит к появлению верхнего и нижнего окаймления. В результате рабочая область экрана, куда выводится информация, хранящаяся в видеобуфере, оказывается окаймленной с четырех сторон. Обычно это окаймление называют бордюром. За цвет бордюра отвечает специальный регистр (см. справочник регистров).

Окаймление экрана

Рабочая область экрана для вывода данных из видеобуфера

Ниже будет дано подробное описание регистров VGA-адаптера. Доступ к большинству регистров осуществляется следующим образом: сначала в порт индекса нужно поместить индекс, а затем через порт данных можно читать или писать данные.

Внешние регистры (цветовой режим)

Это регистры, доступ к которым осуществляется непосредственно через свой порт, а не через индексный регистр. Для монохромного режима адреса портов несколько отличаются. Суть различия такова: там, где для цветового режима в адресе порта рядом с тройкой стоит D, в монохромном - стоит B.

Подробное описание некоторых регистров.

Регистр определения различных режимов работы. MOR - Miscellaneous Output Register.

Запись через порт 3C2H, чтение - 3CCH.

Биты:

0 - выбор адресов ввода-вывода. 0 - выбирается адресное пространство монохромного режима, 1 - адресное пространство цветового режима.

1 - разрешения доступа к видеопамяти. Если бит равен 0, то доступ к видеопамяти будет запрещен.

2,3 - выбор частоты:

0 0 - 640 или 320 столбцов,

0 1 - 720 столбцов,

1 0 - внешний генератор,

1 1 - зарезервировано.

4 - не используется.

5 - выбирает одну из двух графических страниц (64 К).

6,7 - определение скорости сканирования:

0 0 - не используется,

0 1 - 350 линий,

1 0 - 400 линий,

1 1 - 480 линий.

Регистр управления дополнительным устройством. Feature Control Register - FCR.

Доступен для записи через порт 3DAH, для чтения 3CAH.

Не используется, но бит 3 должен быть равен нулю.

Регистр состояния 0. ISRO - Input Status Register.

Доступен для чтения по адресу 3C2H.

Биты:

4 - тип дисплея - цветной или монохромный.

7 - бит прерывания от ЭЛТ. Этот бит устанавливается в начале обратного вертикального хода луча и сбрасывается в конце.

Остальные биты не используются.

Регистр состояния 1. ISR1 - Input Status Register.

Доступен для чтения через порт 3DAH.

Биты:

3 - бит обратного вертикального хода луча. Бит имеет значение 1 в течение обратного вертикального хода луча.

Остальные биты для VGA несущественны.

Регистр разрешения работы системы VGA. VGA_ER - VGA Enable Register.

Управляется через порт 3C3H.

Если бит 0 сброшен, то запрещен доступ к видеопамяти и портам.

Регистры контроллера ЭЛТ.

В цветовом режиме адрес индексного порта 3D4H, адрес порта данных 3D5H. Для монохромного режима адреса этих портов будут соответственно 3B4H, 3B5H. Чтобы

каждый раз не ломать голову, в каком режиме работает программа, можно воспользоваться тем, что адрес индексного регистра расположен по адресу 0000:0463Н.

Мы подробно рассматриваем лишь безопасные регистры ЭЛТ. Неправильное использование некоторых регистров может привести к разрушению дисплея.

Общая длина линии горизонтальной развертки. HTR - Horizontal Total Register. Индекс 0. Регистр определяет число знакомест на одной линии сканирования, включая обратный ход луча и рамку экрана. Число знакомест по горизонтали будет на 5 больше, чем значение, хранящееся в регистре.

Длина отображаемой части горизонтальной развертки. HDER - Horizontal Display Enable End Register. Индекс 1. Задает длину участка горизонтальной развертки. Содержимое регистра на единицу меньше, чем число символов в строке экрана.

Начало импульса гашения луча горизонтальной развертки. SHBR - Start Horizontal Blank Register. Индекс 2. Используется видеоадаптером для определения начала импульса гашения.

Конец импульса гашения луча горизонтальной развертки. EHBR - End Horizontal Blank Register. Индекс 3.

Биты:

0-4 - гашение луча горизонтальной развертки происходит, когда значение равно счетчику длины отображаемой части горизонтальной развертки.

5-6 - влияет на отображение символов в текстовом режиме.

7 - равен единице.

Начало импульса горизонтального обратного хода луча. SHRR - Start Horizontal Retrace Register. Индекс 4. Задает момент начала импульса горизонтального обратного хода луча.

Конец импульса горизонтального обратного хода луча. EHRR - End Horizontal Retrace Register. Индекс 5.

Биты:

0-4 - когда значение битов равно счетчику символов в строке, то обратный горизонтальный ход луча завершается.

5-6 - смещение импульса горизонтального обратного хода луча.

7 - пятый бит регистра конца импульса гашения луча горизонтальной развертки.

Число горизонтальных линий растра. VTR - Vertical Total Register. Индекс 6.

Определяет общее число линий горизонтальной развертки в кадре вертикальной развертки, включая гашение вертикального хода луча и обратный вертикальный ход луча. Биты 9 и 10 содержатся в дополнительном регистре OVR.

Дополнительный регистр. OVR - Overflow Register. Индекс 7.

Биты:

0 - бит 8 VTR.

1 - бит 8 VDER.

2 - бит 8 VRSR.

3 - бит 8 SVBR.

4 - бит 8 LCR.

5 - бит 9 VTR.

6 - бит 9 VDER.

7 - бит 9 VRSR.

Предварительная установка горизонтальной развертки. PRSR - Preset Row Scan Register. Индекс 8.

Позволяет производить плавную прокрутку экрана в текстовом режиме.

Биты:

0-4 - задают для верхней строки текста номер линии в матрице символов, начиная с которой начинаются отображаться символы. Самая верхняя строка текста т.о. отображается не полностью.

5-6 - дополнительные биты регистра горизонтального панорамирования контроллера атрибутов, позволяют сдвигать экран более чем на 8 пикселей.

7 - не используется.

Высота символа текста. MSLR - Max Scan Line Register. Индекс 9.

Используется в текстовом режиме и определяет высоту символа в пикселей.

Биты:

0-4 - высота символов в пикселях.

5 - бит 9 SVBR (индекс 15H).

6 - бит 9 LCR (индекс 18H).

7 - управление двойным сканированием. Если бит равен 1, то в режиме с разрешением по вертикали 200 пикселей для каждой линии растра применяется двойное сканирование, сбрасывание этого бита приводит к увеличению разрешения по вертикали до 400 пикселей.

Начальная линия курсора. CSR - Cursor Start Register. Индекс OAH.

Биты:

0-4 - начальная линия курсора.

5 - если бит установлен, то курсор гаснет.

7-6 - не используются.

Конечная линия курсора. CER - Cursor End Register. Индекс OBH.

Биты:

0-4 - номер последней линии курсора.

5-6 - отклонение курсора.

7 - не используется.

Регистры начального адреса. SAR - Start Adres Register. Старший байт индекс 0CH, младший - 0DH.

Используются для перемещения изображения по экрану и переключению страниц. В регистрах записан адрес видеоданных, которые отображаются в верхнем левом углу экрана. С помощью этих регистров можно быстро переключать отображаемые страницы либо осуществлять перемещение по экрану.

Регистры, определяющие положение курсора на экране. CLR - Cursor Location Register. Старший байт индекс 0EH, младший байт - OFH.

Если курсор находится в точке с координатами X,X то число, хранящееся в указанных регистрах, будет получено по формуле X+5*Y.

Начало обратного вертикального хода луча. VRSR - Vertical Retrace Start Register. Индекс ЮН.

Определяет начало обратного вертикального хода луча. Содержит 10 бит. Лишние биты содержатся в OVR.

Конец обратного вертикального хода луча. VRER - Vertical Retrace End Register. Индекс - 11H.

Доступен для записи.

Биты:

0-3 - когда значение битов будет равно четырем младшим битам счетчика горизонтальных линий, сигнал обратного хода луча будет окончен.

4 - нуль означает сброс вертикального прерывания и переустановку флага незаконченного вертикального прерывания.

5 - при нулевом значении при каждом обратном ходе луча возникает прерывание на линии IRQ2.

6 - если бит равен единице, то во время обратного горизонтального хода луча будет генерироваться пять циклов регенерации видеопамяти вместо 3.

7 - используется для совместимости видеоадаптеров.

Завершение отображения вертикальной развертки. VDER - Vertical Display End Register. Индекс 12H.

Содержит 10 бит. 9 - 10 биты, доступны через OVR.

Содержит число на единицу меньшее, чем количество горизонтальных линий растра.

Логическая ширина экрана. OFR - Offset Register. Индекс 13H.

Может быть использован для отображения большего, чем обычно количества символов в строке.

Положение подчеркивание символа. ULR - Underline Location Register. Индекс 14H.

Биты:

0-4 - определяет положение подчеркивания символа (0-13).

5 - установка этого бита означает, что для каждого знака-места счетчик адреса регенерации будет увеличиваться на 4 вместо 1.

6 - установка этого бита выбирает адресацию видеопамяти по двойным словам.

7 - не используется.

Начало импульса синхронизации. SVBR - Start Vertical Blank Register. Индекс 15H.

Длина 10 бит. Бит 9 расположен в OVR. Бит 10 - в регистре высоты символов (MSLR).

Определяет момент начала гашения луча в процессе вертикальной развертки.

Конец импульса гашения вертикальной развертки. EVBR - End Vertical Blank Register. Индекс 16H.

В момент, когда значение регистра равно счетчику горизонтальных линий, заканчивается сигнал гашения вертикальной развертки.

Управление режимом. MCR - Mode Control Register. Индекс 17H.

Биты:

0 - используется для эмуляции графических режимов CGA.

1 - используется для эмуляции графических режимов Hercules.

2 - может использоваться для увеличения в два раза вертикальной разрешающей способности.

3 - если бит равен нулю, то счетчик адреса регенерации изображения увеличивается на 1 на каждое знакоместо, иначе на каждые два знакоместа.

4 - используется для тестирования.

5 - используется в EGA.

6 - если бит равен нулю - двухбайтовый режим, 1 - однобайтовый.

7 - если бит равен нулю, то горизонтальный и вертикальный обратный ход лучей невозможен.

Регистр сравнения линий. LCR - Line Compare Register. Индекс 18H.

Имеет 10 бит. Бит 9 расположен в OVR. Бит 10 в MSLR.

Когда счетчик горизонтальных линий сканирования достигает значения, записанного в регистре LCR, происходит сброс счетчика адреса регенерируемой видеопамяти в нуль, и экран разбивается на две части. В верхней отображаются данные, на которые указывает регистр начального адреса, а в нижней - данные, находящиеся в начале памяти.

Регистры синхронизатора.

Доступ осуществляется через индексный порт 3C4H и порт данных 3C5H.

Регистр сброса синхронизатора. RR - Reset Register. Индекс 0.

Бит:

0 - бит асинхронного сброса. После записи в него 0, происходит немедленный сброс и остановка синхронизатора. Могут быть потеряны видеоданные.

1 - Бит синхронного сброса. Нулевое значение сбрасывает и останавливает синхронизатор в конце исполняемого цикла.

Регистр режима синхронизации. CMR - Clock Mode Register.

Индекс 1.

Управляет временными циклами синхронизатора. Перед модификацией регистра следует сбросить бит 1 в регистре сброса.

Биты: 0 - задает ширину символов в текстовых монохромных режимах. 5 - если бит равен единице, то экран гаснет и процессор получает монополию на доступ к видеопамяти. Это может несколько ускорить процесс обмена данными между процессором и видеопамятью.

Остальные биты не используются.

Регистр разрешение записи битовой плоскости. CPWE - Color Plane Write Enable. Индекс 2.

Первые четыре бита определяют возможность записи в четыре битовых плоскости. Если бит равен 1, то можно записывать в соответствующую битовую плоскость.

Регистр выбора знакогенератора. CGSR - Character Generator Select Register. Индекс 3.

Видеоадаптер VGA позволяет загрузить до 8 таблиц знакогенератора. Одновременно могут использоваться символы двух таблиц.

Используются биты 0-5.

Бит 3 байта атрибутов, в случае если разрешено использование двух таблиц символов, определяет, какая таблица будет использоваться при отображении символа с данным атрибутом.

Биты 2,3,5 определяют таблицу, если бит 3 байта атрибутов равен 1, биты 0,1,4 - если этот бит равен 0.

Выбор осуществляется согласно таблице:

Биты			
5	3	2	
-	-	-	
4	1	0	
-	-	-	
0	0	0	Таблица 1
0	0	1	Таблица 2
0	1	0	Таблица 3
0	1	1	Таблица 4
1	0	0	Таблица 5
1	0	1	Таблица 6
1	1	0	Таблица 7
1	1	1	Таблица 8

Регистр определения структуры памяти. MMR - Memory Mode Register. Индекс 4.

Биты:

0 - обычно равен 0.

1 - обычно равен 1.

2 - если бит равен 0, то доступ по четным адресам происходит к нулевой битовой плоскости, а по нечетным - к первому (два слоя).

3 - если бит равен 0, то доступ по адресам, кратным 0, к слою 0, кратным 1 - к слою 1, кратным 2 - к слою 2, кратным 3 - к слою 3 (три слоя).

Остальные биты не используются.

Регистры графического контроллера.

Доступ к регистрам осуществляется посредством индексного порта 3CEH и порта данных 3CFH.

Регистр установки/сброса. SRR - Set/Reset Register. Индекс 0.

Используя этот регистр вместе с SRER можно определить данные, размещаемые в битовых плоскостях видеопамяти.

Биты:

0-3 - отвечают за четыре битовых плоскости.

Остальные биты не используются.

Регистр разрешения установки/сброса. SRER - Set/Reset Enable Register. Индекс 1.

Позволяет при операции записи в видеопамять для одних битовых плоскостей использовать данные от процессора, а для других - из регистра SRR.

Биты:

0-3 - отвечают за соответствующие битовые плоскости.

Остальные - резерв.

Если соответствующие биты содержат единицу, то при выполнении операции записи в соответствующие битовые плоскости записывается информация из SRR, в остальные слои записываются данные от процессора (с учетом логических операций, сдвига и содержимого регистра маски).

Возможность использования этого регистра можно осуществить только в нулевом режиме записи (см. MDR).

Регистр сравнения цветов. CCR - Color Compare Register. Индекс 2.

Используется для быстрого поиска на экране пикселя соответствующего цвета.

Биты:

0-3 - искомые величины для соответствующей битовой плоскости.

Остальные биты зарезервированы.

Перед использованием регистра должны быть установлены регистры MDR и CDCR. Более подробно использование этого регистра будет пояснено ниже.

Регистр циклического сдвига и выбора функции. DRFS - Data Rotate and Fuction Select. Индекс 3.

Выполняет две функции:

- циклический сдвиг данных, записываемых процессором в видеопамять,

- выполнение над записываемыми в видеопамять данными и содержимым регистров - защелок некоторых логических операций.

Биты:

0-2 - счетчик сдвига.

3-4 - биты выбора логической функции.

Остальные - зарезервированы.

Операции логического сдвига осуществляются в нулевом режиме записи. Логические операции можно выполнять в нулевом и втором режиме записи. Операция циклического сдвига выполняется до логических операций.

Возможные логические операции: 00 - нет модификации, 01 - логическое "И", 10 - логическое "ИЛИ", 11 - "ИСКЛЮЧАЮЩЕЕ ИЛИ".

Регистр выбора читаемой плоскости. RPSR - Read Plane Select Register. Индекс 4.

Определяет номер битовой плоскости, из которого процессор может читать данные.

0-1 - номер битовой плоскости для чтения.

Регистр режима работы. MDR - Mode Register. Индекс 5.

Биты:

0-1 - режим записи.

Режим 0. Режим непосредственной записи. Процессор имеет доступ к видеопамяти. Возможны операции: установка/сброс, циклический сдвиг, логические операции, использование регистра битовой маски.

Режим 1. Режим использования регистров защелки. При чтении данных происходит запись 8 битов из каждого слоя в регистры защелки. Затем при записи эти данные могут быть записаны в другое место. Используется для копирования.

Режим 2. В этом режиме младшие четыре бита байта данных процессора играют такую же роль, как содержимое регистра установки/сброса в режиме 0.

Режим 3. В данном режиме пиксели модифицируются с помощью объединения значений из регистров защелок и значений в регистре установки/сброса.

2 - не используется.

3 - разрешение режима сравнения цветов. 1 - включение режима сравнения цветов, режим чтения 0 или 1.

4 - четный/нечетный режим. Используется в текстовом режиме.

5 - режим регистра сдвига. Используется в режимах 4 и 5.

6 - управление режимом с 256 цветами.

7 - не используется.

Регистр многоцелевого назначения. MIR - Miscellaneous Register. Индекс 6.

Биты:

0 - бит разрешения графического режима (1).

1 - если бит установлен, то плоскости по 16K объединяются в две плоскости по 32K.

2-3 - установка начального и конечного адресов памяти:

0 0 A000:0000 - B000:FFFF

0 1 AOOOЮOOO - AOOO:FFFF

1 0 B000:0000 - B000:7FFF

1 1 B800:0000 - BOOO:FFFF

Регистр маскирования битовых плоскостей. CDCR - Color Don't Care Register. Индекс 7.

Используется в режиме сравнения цветовых слоев. Если соответствующий бит равен нулю, то при операции сравнения цветов, соответствующий слой в рассмотрение не принимается.

Используются биты 0-3.

Регистр битовой маски. BMR - Bit Mask Register. Индекс 8.

Управляет записью в видеопамять. Если какой-либо бит будет равен нулю, то соответствующий бит будет записываться в видеопамять из регистра защелки.

Регистры контроллера атрибутов.

Доступ осуществляется через один порт ЗСОН, который выполняет роль и индексного регистра, и регистра данных. Для того чтобы перевести порт в исходное (индексное) состояние, следует осуществить чтение из порта 3DAH (ЗВАН для монохромного режима).

Регистры цветовой палитры. CPR - Color Palette Register's.

Индексы 0-15.

Каждый регистр имеет по шесть действующих битов, соответствующих 6 линиям управления дисплея.

Регистр управления дисплеем. MCR - Mode Control Register. Индекс ЮН.

Биты:

0 - содержит нуль для текстовых режимов.

1 - тип атрибутов, для монохромных атрибутов он должен быть равен 1.

2 - используется в монохромных режимах.

3 - управляет 7-м битом в байте атрибутов. Если бит установлен, то 7-й бит определяет мигание, в противном случае - интенсивность.

4 - не используется.

5 - при установке бита запрещено горизонтальное панорамирование стационарной части экрана.

6 - равен нулю с 256 цветами.

1 - выбор источника сигнала для видеовыходов 4 и 5. Если равен нулю, то линии 4 и 5 управляются регистрами палитры, иначе сигналы поступают из битов 0-1 регистра выбора цветов.

Регистр цвета рамки экрана. SBCR - Screen Border Color Register. Индекс 11H.

Биты соответствуют регистрам цветовой палитры.

Регистр разрешения битовой плоскости. CPER - Color Plane Enable Register. Индекс 12H.

Биты:

0-3 - биты разрешения битовых плоскостей (маскирования битовых плоскостей).

4-5 - используются вместе с диагностическими битами регистра состояния.

6-7 - не используются.

Регистр горизонтального панорамирования. HPR - Horizontal Panning Register. Индекс 13H.

0-3 - задают величину горизонтального сдвига.

Регистр выбора цвета. CSR - Color Select Register. Индекс 14H.

Биты:

0 - линия 4.

1 - линия 5.

2 - линия 6.

3 - линия 7.

Регистры цифро-аналогового преобразователя.

Регистр маскирования пикселей. PMR - Pixel Mask Register. Адрес 3C6H.

Регистр состояния ЦАП. DAC_SR - DAC State Register. Адрес 3C7H.

Регистр используется только для чтения.

Первые два бита определяют, доступны или нет регистры цветовой таблицы.

11 - доступны для записи, 00 - доступны для чтения.

Индекс читаемого регистра таблицы цветов. LTRIR - Look-up Table Index Register. Доступен для записи через порт 3C7H.

Данные из регистров таблицы цветов читаются через порт 3C9H как три 6-битовых числа (красный, зеленый, синий). После чтение третьего числа значение индекса автоматически увеличивается на 1. При операции следует запретить прерывания.

Индекс записываемого регистра таблицы цветов. LTWIR - Look-up Table Index Register. Доступен через порт 3C8H.

Запись осуществляется через порт 3C9H аналогично операции чтения.

Регистр данных таблицы цветов. LDTR - Look-up Table Data Register. Доступен через порт 3C9H.

Используется 6 первых бит. Читается и пишется последовательно 3 раза (красный, зеленый, синий).

II. Структура видеопамяти стандартных графических режимов.

Структура видеопамяти для стандартных режимов имеет плоскостной характер. Стандартный объем видеопамяти - 256K, об использовании памяти большего размера речь пойдет ниже. Каждая из четырех битовых плоскостей имеет объем 64K соответственно. Все плоскости соответствуют одному и тому же адресному пространству в памяти и одному и тому же координатному пространству на экране. Каждый пиксель формируется из четырех параллельных бит - по одному из каждой плоскости. Таким образом, в общем случае цвет пикселя формируется из четырех бит, т.е. может иметь значение от 0 до 15. Эта стандартная схема видеопамяти пришла в VGA от старых адаптеров. Она имеет два существенных недостатка:

а) битовое управление пикселями, что достаточно неудобно,

б) для увеличения диапазона цветов необходимо добавлять битовые плоскости.

В VGA-адаптере был добавлен еще один режим 1 3Н-й. Структура видеопамяти в нем линейна. Вся видеопамять разбита на байты, а каждый байт отвечает за свой пиксель, то есть цвет пикселя может меняться в диапазоне 0-255.

Схема работы с видеопамятью в режиме с четырьмя битовыми плоскостями довольно полно изложена в главе 10. Ниже я привожу программу, демонстрирующую все четыре способа записи в видеопамять. Мы работаем здесь с режимом 12Н, которого нет у EGA-адаптера, но, по сути, он нисколько не отличается от режима 640x350, который мы разбирали в главе 10.

```
;программа, использующая различные режимы
;записи в видеопамять
CODE SEGMENT
      ASSUME CS:CODE, DS:CODE
      ORG 100H
BEGIN:
      MOV   AX,0A000H   ;видеопамять
      MOV   ES,AX
      XOR   BX,BX       ;работаем с первым байтом
;режим 640X480, 16
      MOV AX,0012H
      INT ЮН
      CALL  INPUT
;разрешим все битовые плоскости
      MOV DX,3C4H
      MOV AH,0FFH
      MOV AL,2
      OUT DX,AX
;регистр битовой маски
      MOV   DX,3CEH
      MOV   AX,0FF08H ;можно менять все биты
      OUT   DX,AX
```

```
;установка режима чтения/записи ———————————————
      MOV  AX,0005H   ;режим чтения 0/записи-0
      OUT  DX,AX
;установка режима циклического сдвига
      MOV  AX,0003H
      OUT  DX,AX
;регистр разрешения установки/сброса - SRER
      MOV  AX,0001H   ;индекс 1
      OUT  DX,AX
/запись в видеопамять
      MOV  AL,11111111B
      OR   ES:[BX],AL /во все четыре плоскости заноситься
255
/на экране появляется белая полоска
      CALL INPUT
/разрешаем для одной из плоскостей использование данных SRR
      MOV  AH,00000001B /для битовой плоскости О
      MOV  AL,1
      OUT  DX,AX
      MOV  AX,0  /в битовую плоскость 0 будет записан байт О
/в результате цвет каждого из 8 пикселей будет уже не 1111B
;- (белый), а 1110B - желтый
      OUT  DX,AX   ;запись в SRR
      MOV  AL,0FFH
      OR   ES:[BX],AL  /на экране желтая полоска
      CALL INPUT
;теперь попытаемся поставить точку с координатами X=5, Y=5 -
;- шестая строка, цвет точки - 1101B
;маскируем все пиксели, кроме 5-ого ()
      MOV AH,00001000B
      MOV AL,8
      OUT DX,AX
/работаем с регистрами установки/сброса - они обеспечат пра-
;вильный бит в плоскости 1 (бит должен быть равен 0)
      MOV AH,00000010B
      MOV AL,1
      OUT DX,AX
;SRR у нас уже готов
      MOV AL,0FFH
      OR   ES:[BX]+80*5,AL  /на экране точка
      CALL INPUT
/теперь переходим к режиму записи 1 ——————————————
      MOV  AX,0105H   /режим чтения 0/записи-1
      OUT  DX,AX
;перенесем полоску в левом верхнем углу в другое место экрана.
```

```
;Надеюсь Вы понимаете,  что байт передается не через регистр
;AL, а через регистры защелки
      MOV  AL,ES:[BX]
      MOV  ES:[BX]+8000,AL ;сотая строка экрана
      CALL INPUT
;теперь режим записи 2 ------------------------------------------
      MOV  AX,0205H   /режим чтения O/записи-2
      OUT  DX,AX
/поставим точку цвета 1001B с координатами X=5 Y=10
/регистр маски битов у нас уже готов, регистры же установки/
;сброса нас уже не волнуют
      MOV AL,00001001B ;нас интересуют лишь 4 первых бита
      OR   ES:[BX]+80*10,AL
      CALL  INPUT
;режим записи 3    ------------------------------------------
      MOV  AX,0305H    /режим чтения O/записи-3
      OUT  DX,AX
/регистры  установки/сброса
/в данном случае мы лишены возможности выбора битовой
;плоскости, операция будет производиться над всеми плоскостями
/т.о. во все разрешенные пиксели будет записано 0001B - синий
/цвет
      MOV   AL, 0
      MOV  AH, 1
      OUT  DX,AX
/разрешаем все пиксели
      MOV AH,11111111B
      MOV AL,8
      OUT DX,AX
      MOV  AL,0FH  /половина полоски станет синей
      OR   ES:[BX],AL
      CALL INPUT
;возвращаемся в текстовый режим
      CALL  SET_REG
      MOV AX,0003H
      INT ЮН
;возвращаемся в DOS
      MOV AX,4C00H
      INT 21H
/область процедур
;ожидать нажатие клавиши
INPUT PROC
      XOR AX,AX
      INT 16H
      RETN
```

```
INPUT ENDP
;установить значение регистров по умолчанию
SET_REG PROC
;регистр сдвига маски
      MOV  DX,3CEH
      MOV  AX,0FF08H
      OUT  DX,AX
;установка режима чтения/записи
      .MOV  AX,0005H   ;режим чтения 0/записи-0
      OUT  DX,AX
;установка режима циклического сдвига
      MOV  AX,0003H
      OUT  DX,AX
;регистр разрешения установки/сброса - SRER
      MOV  AX,0001H   ;индекс 1
      OUT  DX,AX
      RETN
SET_REG ENDP
CODE ENDS
      END BEGIN
```

Рис. 26.1. Пример использования различных способов записи в видеобуфер.

Если Вы разобрали программы и схемы в главе 10, то без труда поймете программу на Рис. 26.1. Мы же переходим к обсуждению вопроса чтения из видеопамяти. Как Вам уже известно, существует два режима чтения из видеопамяти. В режиме 0 алгоритм чтения тривиален. При помощи регистра RPSR выбираем читаемую плоскость и далее читаем. Например, MOV AL,ES:[BX]. Данный способ хорош, если речь идет о чтении некой области экрана (чтобы записать на диск, например). Чтобы узнать цвет конкретного пикселя, однако, требуется не только вначале считать четыре байта из битовых плоскостей, но и провести анализ соответствующий битов.

Для поиска же конкретного пикселя такой подход неудобен. В этом случае используют режим чтения 1. Здесь значения восьми пикселей, переданных в регистры защелки, сравниваются с четырьмя младшими битами регистра сравнения цветов (CCR). Результат сравнения передается процессору. Когда цвет пикселя совпадет с содержимым регистра CCR, соответствующий бит в возвращаемом байте равен 1, при неравенстве - 0. В процессе считывания участвуют четыре бита регистра CDCR. Они объединяются по "И" со значениями пикселей. Если мы не хотим, чтобы данный регистр влиял на результат, достаточно в его младшие биты поместить 1111b. Данный режим позволяет быстро найти все пиксели заданного цвета. Однако для того чтобы этим методом узнать цвет конкретной точки, необходимо сделать до 16 считываний из памяти.

Одним из интересных графических режимов адаптера VGA является режим 13H. Разрешение в этом режиме составляет 320x200. Структура памяти линейная, т.е. каждый байт ее соответствует одной точке. Значение цвета варьируется в промежутке 0-255. Размер одной (и единственной) страницы этого режима составляет 320x200=64000 байт. Смещение в

видеобуфере байта точки с координатами X,Y вычисляется по элементарной формуле: 320*y+x. Простота и быстрота доступа к пикселям, широкий цветовой диапазон являются причиной частого использования этого режима в графических программах. Ниже представлена простая программа, демонстрирующая чтение-запись пикселей в данном режиме.

```
;пример записи и чтения в режиме 13H
CODE SEGMENT
      ASSUME CS:CODE, DS:CODE
      ORG 100H
BEGIN:
;установка режима
      MOV AX,0013H
      INT 10H
;начнем с нулевого байта
      XOR  BX,BX
      MOV AX,0A000H
      MOV ES,AX
      CALL INPUT
      MOV  CX,320*200 ;количество пикселей на видеостранице
      MOV  AL, 0
L001:
      MOV  ES:[BX],AL
      INC  AL
      INC  BX
      LOOP L001
      CALL INPUT
      XOR  BX,BX
      MOV  CX,64000
;меняем все цвета >=128 на цвет 255
L002:
      CMP  BYTE PTR ES:[BX],128
      JB   NO_CH
      MOV  BYTE PTR ES:[BX],255
NO_CH:
      INC  BX
      LOOP L002
      CALL INPUT
      MOV AX,0003H
      INT 10H
      MOV AH,4CH
      INT 21H
INPUT PROC
      MOV AH,0
      INT 16H
      RETN
```

```
INPUT  ENDP
CODE   ENDS
       END BEGIN
```

Рис. 26.2. Пример работы в режиме 13Н.

Отрицательной стороной режима 13Н является то, что видеопамять состоит всего из одной страницы[67], поэтому возможности мультипликации в этом режиме сильно ограничены. Кроме того, в операциях чтения-записи мы лишены возможности использовать регистры защелки.

III. Нестандартные графические режимы.

Все изложенное выше должно было навести читателя на мысль о недостаточности графических режимов. В частности, хотелось бы иметь видеорежим с 256 цветами и по крайней мере 2 страницами. Простейший расчет позволяет заключить, что это было бы возможно для режима 320x400 для стандартной видеопамяти 256 К. Действительно имеем: 320*400=128000. Это нам как раз и подходит. Т.е. в четыре битовые плоскости по 64 Кб должно поместиться 2 страницы.

Мы рассмотрим два нестандартных графических режима: 320x400 - 256 цветов, 2 - страницы; 640x400 - 16 цветов, 2 страницы.

Начнем с режима 320*400. Режим интересен тем, что необычным для нас способом идет запись в видеопамять. Каждая битовая плоскость отвечает за свой ряд пикселей. В плоскости 0 хранятся цвета пикселей с номерами 0, 4, 8, 12..., в плоскости 1 -сномерами 1, 5, 9, 13..., в плоскости 2 - с номерами 2, 6, 10, 14..., в плоскости 3 - с номерами 3, 7, 11, 15.... Если N - число пикселей в строке, то положение соответствующего байта в видеобуфере определяется по формуле (X+Y*N)/4, номер же плоскости, в которой расположен соответствующий байт, определится по формуле (X+Y*N) MOD 4. При этом поскольку Y*N, естественно, делится на 4, то вычислять надо только остаток X MOD 4. Ниже эти формулы применяются для постановки точки в нужном месте экрана. Отмечу еще, что данные формулы работают и для второго нестандартного режима.

```
CODE   SEGMENT
       ASSUME CS:CODE, DS:CODE
       ORG 100H
BEGIN:
;-----------------------------------
;устанавливаем режим 13H
       MOV   AX,0013H
       INT   10H
       MOV   AX,0A000H
       MOV   ES,AX
```

[67] Вы, наверное, догадались, что это связано с тем, что задействована только одна битовая плоскость.

```
;выбираем регистр синхронизатора с индексом 4
        MOV   DX,3C4H
        MOV   AL,4
        OUT   DX,AL
        INC   DX
        IN    AL,DX
;включаем режим адресации по слоям
        AND   AL,11110111B
;выключаем режим адресации по четным и нечетным адресам
        OR    AL,00000100B
        OUT   DX,AL
;выбираем регистр графического контроллера с индексом 5
        MOV   DX,3CEH
        MOV   AL,5
        OUT   DX,AL
        INC   DX
;выключаем доступ по четным адресам к четным плоскостям,
;а по нечетным адресам - нечетным плоскостям
        IN    AL, DX
        AND   AL, 11101111B
        OUT   DX,AL
        DEC   DX
;выбираем регистр смешанного назначения графического
;контроллера
        MOV   AL, 6
        OUT   DX,AL
        INC   DX
;сбрасываем бит, управляющий сцеплением четных и нечетных
;плоскостей
        IN    AL, DX
        AND   AL,11111101B
        OUT   DX,AL
;разрешаем запись данных во все четыре битовые плоскости,
;записывая число 0FH в регистр разрешения записи в битовую
;плоскость
        MOV   DX,3C4H
        MOV   AL,2
        OUT   DX,AL
        INC   DX
        MOV   AL,00001111B
        OUT   DX,AL
;очищаем видеопамять
;выбор режима 13H очищает только первые 64K.
        XOR   DI,DI
        MOV   AX,DI
```

```
        MOV   CX,0FFFFH
        CLD
        REP   STOSB
;выбираем регистр высоты  символов текста контроллера ЭЛТ
        MOV   DX,3D4H
        MOV   AL, 9
        OUT   DX,AL
        INC   DX
;запрещаем двойное сканирование, т.е. увеличиваем разрешение
;по вертикали в 2 раза
        IN    AL,DX
        AND   AL,01100000B
        OUT   DX,AL
;выбираем регистр положения подчеркивания символа
        DEC   DX
        MOV   AL,14H
        OUT   DX,AL
        INC   DX
;выключаем режим адресации  видеопамяти по двойным словам
        IN    AL,DX
        AND   AL,10111111B
        OUT   DX,AL
;выбираем регистр управления режимом
        DEC   DX
        MOV   AL,17H
        OUT   DX,AL
        INC   DX
;выключаем байтовый режим адресации, в результате получаем
;разделение данных между 4-мя битовыми плоскостями
        IN    AL,DX
        OR    AL,01000000B
        OUT   DX,AL
;------------------------------------
;ставим точку в середину первой страницы
        MOV    CX,160
        MOV    DX,200
        MOV    BL,50
        CALL   POINT
        CALL   INPUT
;ставим точку в середину второй страницы
        MOV    AX,ES
        ADD    AX,800H
        MOV    ES,AX
        MOV    CX,160
        MOV    DX,200
```

```
        MOV   BL, 1
        CALL  POINT
        CALL  INPUT
;переключить страницы, при этом в AL посылается
;номер страницы, умноженный на 4
        MOV   AX,0504H
        INT   10H
        CALL  INPUT
_END:
        MOV  AX,0002H
        INT  10H
        MOV  AH,4CH
        INT  21H
;область процедур
;установить точку в режиме 320*400
;CX - X, DX - Y, BL - цвет
POINT PROC
        MOV  AX,80      ; четверть ширины экрана
        MUL  DX         ; в пикселях
        PUSH CX
        SHR  CX,1
        SHR  CX,1
        ADD  AX,CX
        MOV  DI,AX
        POP  CX
        AND  CL,3       ; остаток от X/4
        MOV  AH,1       ; бит, соответствующий номеру плоскости
        SHL  AH,CL      ; в которую будет занесено значение
цвета
        MOV  DX,03C4H   ; запись в регистр синхронизатора
        MOV  AL,2       ; с индексом 2
        OUT  DX,AX
        MOV  ES:[DI],BL; запись значения цвета в видеопамять
        RETN
POINT ENDP
INPUT PROC
        MOV AH,0
        INT 16H
        RETN
INPUT ENDP
CODE  ENDS
        END BEGIN
```

Рис. 26.3. Пример работы с нестандартным режимом ***320x400.***

Программа на Рис. 26.3 выполняет следующие простые действия: устанавливает графический режим 320x400, ставит точку в середину страницы 0, ставит точку в середину страницы 1, показывает страницу 1. Обращаю ваше внимание на следующий интересный момент: при переключении страницы я вместо 1 засылаю в АН 4. Дело в том, что размер страницы формально равен 8К, тогда как на экране помещается 32К.

Режим 640x400 с 2 страницами памяти и 16 цветами хорош прежде всего тем, что имеет 2 страницы. Работа в этом режиме практически такая же, как в режиме 640x350, поэтому мы только укажем способ перехода в этот режим. Ниже дан фрагмент программы, который переводит адаптер в этот нестандартный режим.

```
mov  ax,000eh       ;режим 640x200 и 16 цветов
int  10h
mov  dx,3d4h
mov  al,9           ;регистр MSLR
out  dx,al
inc  dx
in   al,dx
and  al,01100000b   ;обнуляем 7-й бит
out  dx,al          /отменяя двойное сканирование
```

IV. SVGA и стандарт VESA.

Видеоадаптеры SVGA, к сожалению, не являются стандартом. Для улучшения возможностей своего детища разные фирмы добавляли в него различные средства управления, которые могут значительно отличаться друг от друга. Перед разработчиками стояли две основные задачи: увеличение разрешения экрана и увеличение количества отображаемых цветов. И то и другое требуют увеличения видеопамяти. Заметим при этом, что окно, через которое производится доступ к памяти, остается того же размера: A000H-A000H:FFFFH. О том, как передвигать это окно по видеопамяти, будет сказано ниже.

Вспомним, что в графическом режиме, рассматриваемом в главе 10, на каждый цвет был свой регистр палитры. Меняя регистр палитры, мы имели возможность выбирать 16 цветов из 256 возможных. Соответственно в адаптере таблица цветов формировалась из 256 регистров палитры. Каждый регистр палитры состоит из 6-битных компонент, отвечающих соответственно за интенсивность красной, зеленой, синей компоненты. В результате мы можем получить 256 цветов из набора 64*64*64=262144. В SVGA используется другой принцип формирования цвета, в противном случае пришлось бы многократно увеличивать количество регистров палитры. Здесь используется схема прямого кодирования цвета (Direct Color Mode). Данная схема основана на том, что биты, которые определяют цвет пикселя, разбиваются на три группы, которые определяют интенсивность красного, зеленого, синего цвета. Предположим, на каждый пиксель отводится два байта памяти. В различных режимах из имеющихся 32 бит могут быть выделены различные группы. Рассмотрим, например, ситуацию, когда на каждую компоненту отводится по 5 бит. В результате получаем, что на экране могут существовать одновременно до 32768 цветов (32*32*32=32768, 32 = два в степени 5). В случае если на кодирование цвета отводится 24 бита, получим возможность су-

ществования на экране 256*256*256 = 16777216 цветов. Ниже будет дана таблица режимов SVGA по стандарту VESA.

Функции VESA (VBE - VESA BIOS Extention).

Функции VESA являются хоть каким-то спасением в деле программирования SVGA. Ведь не будете Вы для своей программы писать драйвер для каждого вида адаптера. «Беспредел» в области производства видеоадаптеров дошел уже до того, что стали выпускаться адаптеры, которые не вполне совместимы с адаптерами VGA.

Функции VESA зашиты либо в ПЗУ адаптера, либо поставляются в виде драйвера. При вызове любой функции VESA регистр AH должен содержать 4FH. Номер функции помещается в регистр AL. Если данная реализация поддерживает вызываемую функцию, то в AL возвращается 4FH, если нет, то возвращается другое значение. Если функция завершена успешно, то в AH возвращается 0, если была ошибка, то возвращается 1, и, наконец, если аппаратура видеоадаптера не поддерживает данную функцию, то в AH возвращается 2.

Рассмотрим основные функции VESA.

1. Получить информацию о VBE и видеоадаптере.
   AL = 0
   ES:DI - указатель на буфер в 512 байт (версия VBE 1.2) или буфер 256 (версия 1.2). Первые 4 байта буфера должны содержать строку 'VBE2'.

Содержимое буфера.

Смещение	Размер	Описание
00h	4 байта	Строка "VESA"
04H	2 байта	Номер версии VBE (старший байт старший номер версии, младший - младший номер
06H	4 байта	Указатель на строку, содержащую описание адаптера и VBE
0AH	4 байта	Бит 0 (вер. 1.2) - 1 - ЦАП может работать с данными переменной длины бит 1 (вер. 2.0) - 1 - адаптер не полностью совмести с VGA, бит 2 (вер. 2.0) - 1 - не поддерживает другие функции VBE
0EH	4 байта	Указатель на список возможных режимов. Список состоит из 16-битных величин, в конце стоит FFFFh
12H	2 байта	Только для версии 1.2. Объем видеопамяти в блоках по 64 Кб
		Только для версии 2.0.
14H	2 байта	Дополнительный номер версии.
16H	4 байта	Указатель на строку, содержащую имя фирмы-разработчика
1AH	4 байта	Указатель на строку, содержащую название видеоадаптера

1EH	4 байта	Указатель на строку, содержащую номер версии видеоадаптера
22H	222 байта	Не используется
100H	256 байт	Информация фирмы-производителя

2. Получить информацию о режиме видеоадаптера.
   AL = 1
   CX - номер режима
   ES:DI - указатель на буфер размером 256 байт.

Содержание буфера.

Смещение	Размер	Описание
00H	2 байта	Атрибуты режима бит 0 - 1 - режим доступен бит 1 - 1 - доступна дополнительная информация бит 2 - 1 - поддерживаются функции BIOS бит 3 - 1 - цветовой режим (0 - монохромный) бит 4 - 1 - графический режим бит 5 - 1 - назначение регистров или адресация портов, несовместимая с VGA бит 6 - 1 - нельзя использовать окно A000H:0000 - A000H:FFFF бит 7 - 1 - можно использовать адресацию защищенного режима
02H	1 байт	Атрибуты окна А: бит 0 - доступно, бит 1 - доступно для чтения, бит 2 - доступно для записи,
03H	1 байт	тоже для окна В
04H	2 байта	шаг позиционирования окна в Кб
06H	2 байта	размер окна в Кб
08H	2 байта	начало сегмента окна А
0AH	2 байта	начало сегмента окна В
0CH	4 байта	указатель на функцию перемещения окна
10H	2 байта	количество байт на линию сканирования
		Только для версии VBE 1.2.
12H	2 байта	разрешение по горизонтали

14H	2 байта	разрешение по вертикали
16H	1 байт	ширина символа в пикселях
17H	1 байт	высота символа в пикселях
18H	1 байт	количество битовых плоскостей
19H	1 байт	количество бит на пиксель
1AH	1 байт	количество банков памяти
1BH	1 байт	тип модели памяти: 0 - текстовый режим 1 - структура аналогична CGA 2 - структура аналогична Hercules 3 - видеопамять разделена на 4 слоя (плоскости) 4 - пиксели представлены битами, расположенными последовательно, 5 - режим позволяет отображать 256 цветов, слои не сцеплены 6 - используется схема прямого кодирования
1CH	1 байт	размер банка памяти в Кб
1DH	1 байт	количество доступных страниц видеопамяти без 1
1EH	1 байт	резерв
1FH	1 байт	количество бит для красного компонента
20H	1 байт	положение младшего бита поля
21H	1 байт	количество бит для зеленого компонента
22H	1 байт	положение младшего бита поля
23H	1 байт	количество бит для синего компонента
24H	1 байт	положение младшего бита поля
25H	1 байт	количество запасных бит
26H	1 байт	положение младшего бита запасного поля
27H	1 байт	бит 0 - 1 - если можно программировать регистры ЦАП бит 1 - 1 - если запасное поле можно использовать
		Только для версии VBE 2.0.
28H	4 байта	32-разрядный адрес буфера для использования в защищенном режиме
2CH	4 байта	смещение от начала буфера области памяти, не отображаемой на экране
30H	2 байта	размер неиспользуемой области видеопамяти
32H	206 байт	резерв

3. Установить режим видеоадаптера.
   AL=2h
   BX - номер режима **VESA**

Список графических режимов по стандарту VESA.

Режим	Тип режима	Количество цветов	Разрешение
100H	графический	256	640*400
101H	-	256	640*480
102H	-	16	800*600
103H	-	256	800*600
104H	-	16	1024*768
105H	-	256	1024*768
106H	-	16	1280*1024
107H	-	256	1280*1024
108H	текстовый	16	80*60
109H	-	16	132*25
10AH	-	16	132*43
10BH	-	16	132*50
10CH	-	16	132*60
10DH	графический	32768	320*200
10EH	-	65536	320*200
10FH	-	16777216	320*200
110H	-	32768	640*480
111H	-	65536	640*480
112H	-	16777216	640*480
113H	-	32768	800*600
114H	-	65536	800*600
115H	-	16777216	800*600
116H	-	32768	1024*768
117H	-	65536	1024*768
118H	-	16777216	1024*768
119H	-	32768	1280*1024
ПАН	-	65536	1280*1024

4. Определить текущий графический режим.
   AL = 03H
   Выход: BX - текущий режим.

5. Сохранить/восстановить состояние видеоадаптера.
   AL = 04H
   а) Определить размер буфера состояния.
      DL = 0
      CX - параметр сохранения
         бит 0 - состояние видеоадаптера
         бит 1 - состояние переменных видеофункций
         бит 2 - состояние регистров ЦАП
         бит 3 - состояние регистров SVGA
   Выход:
      BX - размер буфера
   б) Сохранить состояние видеоадаптера
      DL = 1
      ES:BX - указатель на буфер
      CX - параметр сохранения
   в) Восстановить состояние видеоадаптера
      DL = 2
      ES:BX - указатель на буфер
      CX - параметр сохранения

6. Переместить окно видеопамяти.
   AL = 05
   BH = 0
   BL - номер окна (0,1 - A,B)
   DX - адрес окна в единицах шага.
   Шаг кратен размеру окна.

7. Определить адрес окна.
   AL = 5
   BH = 1
   BL - номер окна
   На выходе:
   DX - адрес окна в единицах шага.

Далее в главе будут приведены несколько примеров работы с режимами SVGA. Следующая программа устанавливает режим 103H (800*600*256) и заполняет экран цветными точками. Обращаю Ваше внимание на следующие особенности:

а) объем памяти, требуемый для одной видеостраницы, превосходит 256 Кб - 800*600=480000. Т.о. мы переходим тот рубеж, который когда-то был определен разработчиками VGA-адаптеров. Для того чтобы перейти этот рубеж, мы исполь-

зуем средство VESA BIOS. Других способов автор не знает. К сожалению, утверждение некоторых авторов, что это можно сделать посредством установки окна в пределах A000H:0000 - B000H:FFFFH (см. регистр MIR) не подтверждается. Их программа не стала работать ни на одном из проверяемых компьютеров. В то же время анализ предложенной ими программы не подтверждает наличие в ее тексте опечатки. Следовательно, увы, и не существует универсального способа перейти вышеуказанную границу, кроме как использовать средства VESA [68].

б) структура памяти в устанавливаемом нами режиме аналогична структуре режима 320*400*256, который мы устанавливали ранее. Каждая битовая плоскость отвечает за свой ряд пикселей (см. комментарий к программе на Рис. 26.3).

в) в процедуре постановки точки мы вынуждены проверять, не превосходит ли число (800*Y+X)/4 OFFFFH. Как только это происходит, мы передвигаем адресное окно на один шаг. Этого достаточно, чтобы охватить вторую половину экрана.

г) переменная _SI введена, чтобы не вызывать лишний раз прерывание 1OH. Это бы еще более замедлило вывод информации на экран. Другим фактором, который замедляет постановку точки, является необходимость вызова процедуры - постоянное выполнение команд CALL и RETN является серьезным замедляющим фактором. Но о быстром выводе графической информации речь еще впереди.

В качестве упражнения автор предложил бы читателю переработать программу так, чтобы она делала то же самое для режима 101H (640*480*256). Здесь также придется использовать функции VESA, т.к. и здесь предел в 256 Кб оказывается превзойденным. В этой связи замечу, что с режимом 640*400*256 можно работать без использования средств VESA (кроме установки режима).

```
.286
CODE  SEGMENT
      ASSUME CS:CODE, DS:CODE
      ORG 100H
BEGIN:
;-----------------------------------
;устанавливаем режим 103H
      MOV AX,4F02H  ;пытаемся установить режим 800*600*256
      MOV BX,103H
      INT ЮН
      CMP AX,004FH
      JZ   _OK
      JMP  _END     ;режим не удалось установить
_OK:
      MOV  AX,0A000H
      MOV  ES,AX
```

[68] Остается правда малая вероятность, что указанная программа будет работать с VGA-адаптерами с памятью, превосходящей 256 Кб. Те же SVGA-адаптеры, на которых я проверял ее, были не совсем совместимыми с VGA.

```
;*********************************************************
;выбираем регистр синхронизатора с индексом 4
;с помощью которого можно определить структуру  памяти
      MOV   DX,3C4H
      MOV   AL,4
      OUT   DX,AL
      INC   DX
      IN    AL,DX
;включаем режим адресации по слоям
      AND   AL,11110111B
;выключаем режим адресации по четным и нечетным адресам
      OR    AL,00000100B
      OUT   DX,AL
;*********************************************************
;разрешаем запись данных во все  четыре  битовые  плоскости,
;записывая число 0FH в регистр разрешения  записи в битовую
;плоскость (адрес 3C4H, индекс 2).
      MOV   DX,3C4H
      MOV   AX,0FF02H
      OUT   DX,AX
;*********************************************************
;заполняем экран разноцветными точками
      MOV CX,600
LOO2:
      PUSH CX
      INC   Y
      MOV   X,0
      MOV   CX,800
LOO1:
      PUSH CX
      MOV    CX,X
      MOV    DX,Y
      MOV    BL,COLOR
      CALL   POINT
      INC    COLOR
      INC    X
      POP    CX
      LOOP   LOO1
      POP CX
      LOOP   LOO2
      CALL   INPUT
;*********************************************************
_END:
      MOV   AX,0002H
      INT   10H
```

```
      MOV  AH,4CH
      INT  21H
;область процедур
;установить точку в режиме 800*600*256
;CX - X, DX - Y, BL - цвет
POINT PROC
      MOV  AX,200     ; четверть ширины экрана
      MUL  DX         ; в пикселях
      MOV  DI,CX
      SHR  CX,2
      ADD  AX,CX      ;X/4+800*y/4 -> DX:AX
      ADC  DX,0
      CMP  DX,0       ;проверяем, не передвинуть ли окно
      JNZ  _0
      MOV  SI,0
      CMP  _SI,SI
      JZ   _1
      CALL WIN_MOVE
      JMP SHORT _1
_0:
      MOV  SI,1
      CMP  _SI,SI ;проверяем, может быть, окно не передвигать
      JZ   _1
      CALL WIN_MOVE
_1:
      MOV  SI,AX
      MOV  CX,DI
      AND  CL,3       ; остаток от X/4
      MOV  AH,1       ; бит, соответствующий номеру плоскости
      SHL  AH,CL      ; в которую будет занесено значение
цвета
      MOV  DX,03C4H   ; запись в регистр синхронизатора
      MOV  AL,2       ; с индексом 2
      OUT  DX,AX
      MOV  ES:[SI],BL; запись значение цвета в видеопамять
      RETN
POINT ENDP
;процедура ожидания нажатия клавиши
INPUT PROC
      MOV AH,0
      INT 16H
      RETN
INPUT ENDP
```

```
;процедура передвижения окна видимости на N шагов
;количество шагов помещается в SI, шаг реально
;соответствует размеру окна
WIN_MOVE PROC
       MOV   _SI,SI
       PUSH AX
       PUSH BX
       PUSH DX
       MOV   AH,4FH
       MOV   AL,05
       MOV   BH,0
       MOV   BL,0
       MOV   DX, SI
       INT   10H
       POP   DX
       POP   BX
       POP   AX
       RETN
WIN_MOVE ENDP
;область переменных
;текущие координаты точки
X      DW  0
Y      DW  0
;текущий цвет точки
COLOR DB  0
;хранится текущее положение окна
_SI    DW  0
CODE   ENDS
       END BEGIN
```

*Рис. 26.4. Пример работы с режимом 800*600*256.*

В качестве еще одного примера рассмотрим режим 10DH по стандарту VESA. Этот режим имеет разрешение 320*200 и 32768 цветов. Формирование цвета происходит путем смешивания трех цветов: красного, зеленого и синего. На каждый цвет отводится по 5 байт. Это как раз случай прямого кодирования. Цвет формируется непосредственно занесением байт в память. При этом на каждую точку отводится по два байта. Пятнадцать первых бит и составляют цветовую гамму точки. В результате мы получаем, что для экранной памяти требуется 320*200*2=128000 байт. Точка формируется двумя соседними байтами памяти. Для того чтобы работать со второй половиной экрана нам придется передвинуть окно так, как мы это делали в предыдущем примере.

```
;пример записи и чтения в режиме 13H
CODE SEGMENT
       ASSUME CS:CODE, DS:CODE
       ORG 100H
```

```
BEGIN:
;установка режима
;устанавливаем режим 10DH
      MOV AX,4F02H ;пытаемся установить режим 320*200*32768
      MOV BX,10DH
      INT ЮН
      CMP AX,004FH
      JZ   _OK      ;режим не удалось установить
      JMP  _END
_OK:
      XOR  BX,BX
      MOV AX,0A000H
      MOV ES,AX
;вначале поставим точку красного цвета
      CALL  INPUT
      MOV BX,20000
      MOV  AL,00000000B
      MOV  ES:[BX],AL
      INC BX
      CALL  INPUT
      MOV  AL,11111100B  ;старшие 5 бит и определяют красный
      MOV ES:[BX],AL      ;цвет
      CALL INPUT
/заполним пол-экрана разноцветными точками
      MOV  CX,320*200 /количество пикселей на видеостранице
      MOV  AL,0
      MOV  BX,0
LOO1:
      MOV  ES:[BX],AL
      INC  AL
      INC  BX
      MOV  ES:[BX],AL
      INC  BX
      LOOP LOO1
;передвигаем окно
      MOV  AH,4FH
      MOV  AL, 5
      MOV  BX,0
      MOV  DX, 1
      INT  10H
      CALL INPUT
;заполняем разноцветными точками вторую половину экрана
      MOV  CX,320*200 ;количество пикселей на видеостранице
      MOV  AL,0
      MOV  BX,0
```

```
L002 :
        MOV   ES:[BX],AL
        INC   AL
        INC   BX
        MOV   ES:[BX],AL
        INC   BX
        LOOP  L002
        XOR   BX,BX
        CALL  INPUT
_END:
        MOV AX,0003H
        INT 10H
        MOV AH,4CH
        INT 21H
INPUT  PROC
        MOV  AH,0
        INT 16H
        RETN
INPUT  ENDP
CODE  ENDS
        END BEGIN
```

Рис. 26.5. Пример работы с режимом ***10DH.***

Приложение 1. Система команд микропроцессора.

В приложении приводятся все команды микропроцессоров 8088/8086. Особо рассматриваются команды, не описанные ранее. С командами других микропроцессоров семейства можно познакомиться в главах 5, 20.

Коррекция арифметических действий с неупакованными BCD-числами.

Мнемоника	Описание	Число тактов	Число байт
AAA	Коррекция неупакованных BCD-чисел после сложения.	4	1
AAD	Коррекция неупакованных BCD-чисел до деления.	60	2
AAM	Коррекция неупакованных BCD-чисел после умножения.	83	2
AAS	Коррекция неупакованных BCD-чисел после вычитания.	4	1

Неупакованные BCD-числа или числа в ASCII-формате представляют собой последовательность байт, каждый из которых соответствует одному разряду числа в десятичном представлении (цифра или ее ASCII-код). Ниже представлена программа, показывающая, как можно складывать или вычитать числа в ASCII-формате. Числа представлены строками P1 и P2. Результаты сложения и вычитания помещаются соответственно в строки SUM и RAZ. Действия производятся в предположении, что результат является двухразрядным числом.

```
CODE  SEGMENT
      ORG  100H
      ASSUME CS:CODE
BEGIN;
;вычисление и вывод суммы
      MOV AL,CS:P1+1
      ADD AL,CS:P2+1
      AAA                  ;коррекция сложения младших разрядов
      MOV CS:SUM+1,AL
      MOV AL,CS:P1
      ADC AL,CS:P2
      AAA                  ;коррекция сложения старших разрядов
      MOV CS:SUM,AL
;получение строки
      ADD BYTE PTR CS:SUM,48
      ADD BYTE PTR CS:SUM+1,48
      MOV DX,OFFSET CS:SUM
      MOV AH,9
      INT 21H
```

```
;вычисление и вывод разности
       MOV AL,CS:P1+1
       SUB AL,CS:P2+1
       AAS              ;коррекция вычитания младших разрядов
       MOV CS:RAZ+1,AL
       MOV AL,CS:P1
       SBB AL,CS:P2
       AAS              ;коррекция вычитания младших разрядов
       MOV CS:RAZ,AL
;получение строки
       ADD BYTE PTR CS:RAZ,48
       ADD BYTE PTR CS:RAZ+1,48
       MOV DX,OFFSET CS:RAZ
       MOV AH,9
       INT 21H
       RET
P1     DB '39'
P2     DB '12'
SUM DB '  ',13,10,'$'
RAZ DB '  ',13,10,'$'
CODE ENDS
       END BEGIN
```

Команда AAM облегчает умножение чисел, представленных в ASCII-формате. Эта команда делит содержимое AL на 10, помещая результат в AH, а остаток в AL. Ниже представлен пример использования этой команды для умножения двух чисел.

```
CODE SEGMENT
       ORG 100H
       ASSUME CS:CODE
BEGIN:
;умножаем P1 на P2 предположения P1*P2<100
       MOV  CL,CS:P1+1
       MOV  AL,CS:P2
       SUB  AL,48
       SUB  CL,48
       MUL  CL        ;3*7
       AAM
       PUSH AX
       MOV  CL,CS:P1
       MOV  AL,CS:P2
       SUB  AL,48
       SUB  CL,48
       MUL  CL        ;3*2
       MOV  AH,AL
       XOR  AL,AL
       POP  CX
       ADD  AL,CL     ;1+0
```

```
        ADC   AH,CH   ;6+2
        OR    AX,3030H     ;перевод в ASCII-формат
        MOV   CS:MUL1,AH
        MOV   CS:MUL1+1,AL
;выводим результат
        LEA   DX,CS:MUL1
        MOV   AH, 9
        INT   21H
        RET
P1      DB  '27'
P2      DB  '3'
MUL1    DB  '   ',13,10,'$'
CODE  ENDS
        END BEGIN
```

Аналогично для деления в ASCII-формате используется команда AAD, которая загружает в AL AL+AH* 10.

Коррекция арифметических действий с упакованными BCD-числами.

Мнемоника	Описание	Число тактов	Число байт
DAA	Коррекция упакованных BCD-чисел после сложения.	4	1
DAS	Коррекция упакованных BCD-чисел после вычитания.	4	1

Команды арифметических действий.

Мнемоника	Описание	Число тактов	Число байт
ADD	Сложить		
	регистр - регистр	3	2
	память - регистр	9+EA	2-4
	регистр - память	16+EA	2-4
	непосредственный операнд - регистр	4	3-4
	непосредственный операнд - память	17+EA	3-6
	непосредственный операнд - аккумулятор	4	2-3
ADC	Сложить с переносом		
	регистр - регистр	3	2
	память - регистр	9+EA	2-4
	регистр - память	16+EA	2-4
	непосредственный операнд - регистр	4	3-4
	непосредственный операнд - память	17+EA	3-6
	непосредственный операнд - аккумулятор	4	2-3

SUB	Вычесть		
	регистр - регистр	3	2
	память - регистр	9+EA	2-4
	регистр - память	16+EA	2-4
	непосредственный операнд - регистр	4	3-4
	непосредственный операнд - память	17+EA	3-6
	непосредственный операнд - аккумулятор	4	2-3
SBB	Вычесть с переносом		
	регистр - регистр	3	2
	память - регистр	9+EA	2-4
	регистр - память	16+EA	2-4
	непосредственный операнд - регистр	4	3-4
	непосредственный операнд - память	17+EA	3-6
	непосредственный операнд - аккумулятор	4	2-3
NEG	**Изменить знак**		
	регистр	3	2
	память	16+EA	2-4
DEC	Декремент на 1		
	16-битный регистр	2	1
	8-битный регистр	3	2
	память	15+EA	2-4
INC	Инкремент на 1		
	16-битный регистр	2	1
	8-битный регистр	3	2
	память	15+EA	2-4
MUL	Умножить без знака		
	8-битный регистр	70-77	2
	16-битный регистр	118-133	2
	8-битная память	(76-83)+	2-4
	16-битная память	EA (124-139)	2-4
		+EA	
IMUL	Умножить со знаком		
	8-битный регистр	80-98	2
	16-битный регистр	128-154	2
	8-битная память	(86-104)+	2-4
	16-битная память	EA (134-160)	2-4
		+EA	
DIV	Деление без знака		
	8-битный регистр	80-90	2
	16-битная память	144-162	2
	16-битный регистр	(86-96)+	2-4
	8-битная память	EA (150-168)	2-4
		+EA	

ГОР/	Деление со знаком 8-битный регистр 16-битный регистр 8-битная память 16-битная память	 101-112 165-184 (107-118) +EA (171-190) +EA	 2 2 2-4 2-4
CBW	Преобразовать байт в слово	2	1
CWD	Преобразование слова в двойное слово	5	1

Поясню некоторые обозначения, появившиеся в данной таблице. EA - означает добавку ко времени выполнения за счет необходимого обмена с памятью. Например, команда INC MEM, где MEM — некоторая ячейка памяти, предполагает вычисление эффективного адреса, соответствующего данной ячейке. В данном случае адресация была прямой и требует для вычисления эффективного адреса 6 тактов. Команда же INC [BX] предполагает косвенную адресацию. Для вычисления эффективного адреса при такой адресации требуется 5 тактов. И т.д. Длина команды может варьироваться. Например, запись 2-4 означает, что команда может быть длиной 2 или 4 байта: команда ADD BX,MEM длиной 4 байта, команда ADD BX,[DI] - длиной 2 байта и т.п.

Команды передачи данных.

Мнемоника	Описание	Число тактов	Число байт
IN	Вывести из порта ввода-вывода фиксированный порт: IN AL,61H переменный порт: MOV DX,61H/IN AL,DX	 10 8	 2 1
OUT	Вывести из порта ввода-вывода фиксированный порт: OUT 61H,AL переменный порт: MOV DX,61H/OUT DX,AL	 10 8	 2 1
MOV	Переслать аккумулятор - память память - аккумулятор регистр - регистр память - регистр регистр - память непосредственный операнд - регистр непосредственный операнд - память регистр - сегментный регистр (кроме CS) память - сегментный регистр (кроме CS) сегментный регистр - регистр сегментный регистр - память	 10 10 2 8+EA 9+EA 4 10+EA 2 8+EA 2 9+EA	 3 3 2 2-4 2-4 2-3 3-6 2 2-4 2 2-4

LEA	Загрузить эффективный адрес (смещение)	2+EA	2-4
LDS	Загрузить регистр вместе с DS (указатель)	16+EA	2-4
LES	Загрузить регистр вместе с ES (указатель)	16+EA	2-4
LAHF	Загрузить AH из флажков	4	1
SAHF	Запомнить AH во флажках	4	1
LODSB	Загрузить цепочку байт		
LODSW	Загрузить цепочку слов без повторения с повторением	 12 9+13/пов.	 1 1
MOVSB	Переслать цепочку байт		
MOVSW	Переслать цепочку слов без повторения с повторением	 18 9+17/пов.	 1 1
SCASB	Сканировать цепочку байт		
SCASW	Сканировать цепочку слов без повторения с повторением	 15 9+15/пов.	 1 1
STOSB	Запомнить цепочку байт		
STOSW	Запомнить цепочку слов без повторения с повторением	 И 9+10/пов.	 1 1
POPF	Извлечь флажки из стека	8	1
PUSHF	Включить флажки в стек	8	1
ГОР	Извлечь слово из стека в регистр в сегментный регистр (кроме CS) в память	 8 8 17+EA	 1 1 2-4
PUSH	Включить слово в стек из регистра из сегментного регистра из памяти	 11 10 16+EA	 2-4 1 1
XLATB	Загрузить AL из таблицы	11	1
XCHG	Обменять регистр - аккумулятор регистр - память регистр - регистр	 3 11+EA 4	 2-4 1 1

Запись типа 9+10/пов. означает, что возможны два варианта работы команды. Если повторение осуществлялось, тогда время будет 9+пов. Если же число повторений было 0 (CX=0), то время будет равно 9+10.

Логические команды и команды манипуляции битами.

Мнемоника	Описание	Число тактов	Число байт
AND	Побитовое "И"		
	регистр - регистр	3	2
	память - регистр	9+EA	2-4
	регистр - память	16+EA	2-4
	непосредственный операнд - регистр	4	3-4
	непосредственный операнд - память	17+EA	3-6
	непосредственный операнд - аккумулятор	4	2-3
OR	Побитовое "ИЛИ"		
	регистр - регистр	3	2
	память - регистр	9+EA	2-4
	регистр - память	16+EA	2-4
	непосредственный операнд - регистр	4	3-4
	непосредственный операнд - память	17+EA	3-6
	непосредственный операнд - аккумулятор	4	2-3
XOR	Побитовое исключающее "ИЛИ"		
	регистр - регистр	3	2
	память - регистр	9+EA	2-4
	регистр - память	16+EA	2-4
	непосредственный операнд - регистр	4	3-4
	непосредственный операнд - память	17+EA	3-6
	непосредственный операнд - аккумулятор	4	2-3
NOT	Отрицание (инвертирование)		
	регистр	3	2
	память	16+EA	2-4
RCL	Циклически сдвинуть влево через бит переноса		
	регистр (на один бит)	2	2
	регистр (переменный сдвиг)	8+4/бит	2
	память (на один бит)	15+EA	2-4
	память (переменный сдвиг)	20+EA+	2-4
RCR	Циклически сдвинуть вправо через бит переноса		
	регистр (на один бит)	2	2
	регистр (переменный сдвиг)	8+4/бит	2
	память (на один бит)	15+EA	2-4
	память (переменный сдвиг)	20+EA+	2-4

ROL	Циклически сдвинуть влево		
	регистр (на один бит)	2	2
	регистр (переменный сдвиг)	8+4/бит	2
	память (на один бит)	15+EA	2-4
	память (переменный сдвиг)	20+EA+	2-4
ROR	Циклически сдвинуть вправо		
	регистр (на один бит)	2	2
	регистр (переменный сдвиг)	8+4/бит	2
	память (на один бит)	15+EA	2-4
	память (переменный сдвиг)	20+EA+	2-4
		4/бит	
		4/бит	
SAL	Сдвинуть влево с учетом знака		
	регистр (на один бит)	2	2
	регистр (переменный сдвиг)	8+4/бит	2
	память (на один бит)	15+EA	2-4
	память (переменный сдвиг)	20+EA+	2-4
SHL	Сдвинуть влево без учета знака (логически)		
	регистр (на один бит)	2	2
	регистр (переменный сдвиг)	8+4/бит	2
	память (на один бит)	15+EA	2-4
	память (переменный сдвиг)	20+EA+	2-4
SAR	Сдвинуть вправо с учетом знака		
	регистр (на один бит)	2	2
	регистр (переменный сдвиг)	8+4/бит	2
	память (на один бит)	15+EA	2-4
	память (переменный сдвиг)	20+EA+	2-4
SHR	Сдвинуть вправо без учета знака (логически)		
	регистр (на один бит)	2	2
	регистр (переменный сдвиг)	8+4/бит	2
	память (на один бит)	15+EA	2-4
	память (переменный сдвиг)	20+EA+	2-4
		4/бит	

Команды сравнения.

Мнемоника	Описание	Число тактов	Число байт
CMP	Сравнить		
	регистр - регистр	3	2
	память - регистр	9+EA	2-4
	регистр - память	9+EA	2-4
	непосредственный операнд - регистр	4	3-4
	непосредственный операнд - память	10+EA	3-6
	непосредственный операнд - аккумулятор	4	2-3
CMPSB	Сравнить цепочку байт		
CMPSW	Сравнить цепочку слов		
	без повторения	22	1
	с повторением	9+22/пов.	1
TEST	Проверить		
	регистр - регистр	3	2
	память - регистр	9+EA	2-4
	непосредственный операнд - аккумулятор	4	2-3
	непосредственный операнд - регистр	4	3-4
	непосредственный операнд - память	17+EA	3-6

Данные команды удобны тем, что действуют на флажки, но не на содержимое регистров.

Команды переходов и повторений.

Мнемоника	Описание	Число тактов	Число байт
CALL	Вызов процедуры		
	внутрисегментный прямой	19	3
	внутрисегментный косвенный через регистр	16	2
	внутрисегментный косвенный через память	21+EA	2-4
	межсегментный прямой	28	5
	межсегментный косвенный	37+EA	2-4
ЮТ	Вызов прерывания		
	тип=3	52	1
	тип=/3	51	2
INTO	Прерывание при переполнении		
	прерывание есть	54	1
	прерывания нет	4	1

IRET	Возвратиться из прерывания	24	1
RET	Возврат из процедуры внутрисегментный внутрисегментный с константой межсегментный межсегментный с константой	 8 12 18 17	 1 3 1 3
REP	Повторить цепочечную операцию	2	1
REPE	Повторить операцию, пока равно	2	1
REPZ	Повторить операцию, пока нуль	2	1
REPNE	Повторить операцию, пока не равно	2	1
REPNZ	Повторить операцию, пока не нуль	2	1
LOOP	Зациклить	17/5	2
LOOPE	Зациклить, если равно	18/6	2
LOOPZ	Зациклить, если нуль	18/6	2
LOOPNE	Зациклить, если не равно	19/5	2
LOOPNZ	Зациклить, если не нуль	19/5	2
JMP	Безусловный переход внутрисегментный прямой короткий (SHORT) внутрисегментный прямой (NEAR) межсегментный прямой внутрисегментный косвенный через память внутрисегментный косвенный через регистр	 15 15 15 15+EA 11	 2 3 5 2-4 2
JCXZ	Перейти, если CX=0	18/6	2
JA/JNBE	Перейти, если выше	16/4	2
JAE/JNB	Перейти если выше или равно	16/4	2
JB/JNAE	Перейти, если ниже	16/4	2
JBE/JNA	Перейти, если не ниже	16/4	2
JE/JZ	Перейти, если равно	16/4	2
JG/JNLE	Перейти, если больше	16/4	2
JGE/JNL	Перейти, если больше или равно	16/4	2
JL/JNGE	Перейти, если меньше	16/4	2
JLE/JNG	Перейти, если меньше или равно	16/4	2
JC	Перейти, если перенос	16/4	2
JNC	Перейти, если переноса нет	16/4	2

Через черточку здесь указаны идентичные команды (JA/JNBE). Запись же типа 16/4 означает, что команда выполняется за четыре такта при выполнении условия и за шестнадцать в противном случае.

Команды управления.

Мнемоника	Описание	Число тактов	Число байт
CLC	Сбросить флажок переноса	2	1
CLD	Сбросить флажок направления	2	1
CLI	Сбросить флажок прерывания	2	1
CMC	Инвертировать флажок переноса	2	1
ESC	Переключиться на сопроцессор регистр память	 2 8+EA	 2 2-4
HLT	Остановить	2	1
NOP	Холостая команда	3	1
STC	Установить флажок переноса	2	1
STD	Установить флажок направления	2	1
STI	Установить флажок прерывания	2	1
WAIT	Ожидать активного сигнала	3+5	1

Приложение 2. Знаковые числа.

Мы почитаем всех нулями
А единицами - себя.

А. С. Пушкин
Евгений Онегин.

Здесь кратко излагаются принципы работы со знаковыми числами. Идея введения в машинный язык отрицательных чисел очень проста. Она позволяет трактовать одно и тоже число и как беззнаковое и как знаковое. Причем действия сложения и вычитания будут выполняться вне зависимости от того, как мы это число трактуем. Программируя на языке ассемблера, Вы сами должны помнить, как трактуется то или иное число.

Попробуем самостоятельно построить систему знаковых чисел. Было бы разумно признаком отрицательного числа считать наличие старшего бита в нем. Соответственно отсутствие бита будет означать, что знак у числа положительный.

Рассмотрим однобайтовые числа. Пусть 00000001В будет представлять 1. По определению -1 должна удовлетворять соотношению: 1+(-1)=0. Но таким числом является 1111111В (старший бит равен 1). Далее, число 2 будет представлено как 00000010В. -2 должно получаться из -1 как (-1)-1. Получим 11111111В-00000001В=11111110В. Таким образом, -2 представляется в двоичном виде как 11111110В. При этом, разумеется, выполнится соотношение 2+(-2)=0. Данный процесс можно продолжать и дальше. При этом получим следующий ряд однобайтовых знаковых чисел:

```
 0  - 00000000B
 1  - 00000001B
 2  - 00000010B
 3  - 00000011B
 .
 .
 .
-3  - 11111101B
-2  - 11111110B
-1  - 11111111B
```

Мы видим, что в одном байте помещаются отрицательные числа от -128 до 127: 127 - 01111111В, -128- 10000000В.

Аналогичные выкладки можно проделать и с двухбайтовыми числами. В результате получим ряд отрицательных чисел от -32768 до 32767.

Как уже было сказано, как сложение, так и вычитание будут правильно выполняться и над числами со знаком и над беззнаковыми числами - все дело в Вашей интерпретации. В случае же умножения и деления существуют отдельные команды для знаковых чисел (см. главу 4 и Приложение 1). Существуют и специальные команды работы со знаковыми числами: смена знака NEG, преобразовать байт в слово с учетом знака CBW, команды сдвига с учетом знаков.

С появлением 386-х микропроцессоров в обиход вошла 32-битная арифметика. Ничего принципиально нового она в наши рассуждения не вносит. Вы лишь должны внимательно отслеживать, с числами какой разрядности имеет дело данная команда.

Приложение 3. Директивы и команды макроассемблера.

Как много, однако, существует
такого, в чем я не нуждаюсь.

Сократ.

В данном приложении приводятся основные команды и директивы макроассемблера. Некоторые из них я использую в своей книге, другие приведены здесь для ознакомления. Надо сказать, что я придерживаюсь того мнения, что многие макросредства, несомненно, облегчая процесс программирования, усложняют понимание сути программы. В программах данной книги Вы не найдете ни макроопределений, ни условных конструкций, ни других подобных вещей. Впрочем, это дело вкуса и привычки.

Директивы SEGMENT - ENDS.

Служат для выделения в программе частей с единой адресацией относительно начала сегмента.

Структура:

```
имя SEGMENT [счетный т.][атрибут][комбинированный т.][размер]['класс']
.
.
имя ENDS
```

Счетный тип - BYTE, WORD, PARA (16 байт), PAGE (256 байт). Определяет тип выравнивания сегмента.

Атрибут - может иметь значение READONLY. Если какая-либо команда модифицирует такой сегмент, то при ассемблировании будет сгенерирована ошибка.

Комбинированный тип:

public - конкатенация всех сегментов, имеющих одинаковое имя, в один, вся адресация пересчитывается относительно этого сегмента.

stack - конкатенация всех сегментов с одинаковым именем в один, на его начало будет указывать SS, SP - на конец сегмента.

common - соединение всех сегментов, имеющих одно имя, в один сегмент, размер которого будет равен наибольшему. Поскольку сегменты будут перекрываться, то и содержимое сегментов будет перекрываться также.

memory - совпадает с common.

at адрес - создание сегмента-шаблона относительно заданного адреса.

Размер: USE16 - 16-битный сегмент, USE32 - 32-битный сегмент, FLAT - модель, когда вся память состоит из одного сегмента.

Тип класса: сегменты с одинаковым именем класса загружаются один за другим.

Директива GROUP.

Данная директива собирает однотипные сегменты так, чтобы адресация в них была единой.

имя GROUP имя1,имя2, ...

Ниже представлена простая программа, где используется директива GROUP. Несмотря на то, что она состоит из двух сегментов, ее EXE-модуль может быть преобразован к COM-формату.

```
GR GROUP CODE,DAT1
CODE SEGMENT BYTE
      ASSUME CS:CODE
      ORG 100H
BEGIN:
      MOV DX,OFFSET GR:STRING
      MOV AH, 9
      INT 21H
      RET
CODE ENDS
DAT1 SEGMENT BYTE
STRING DB 'Проверка.',13,10, '$'
DAT1 ENDS
      END BEGIN
```

Директива INCLUDE.

Директива имеет формат: INCLUDE имя_файла. Посредством нее в данном месте программы подключается ассемблерный текст из другого файла. Посредством этой директивы можно собирать программу из нескольких отдельных текстов, что несколько облегчает разработку больших программ. Директива INCLUDE допускает вложение.

Директивы LENGTH, SIZE, TYPE.

LENGTH - возвращает число элементов, определенных операндом DUP: DW 64 DUP (?), DD 10DUP(O) и т.п.

TYPE - возвращает число байт, соответствующих определению указанной переменной.

SIZE=TYPE*LENGTH.

Директива NAME.

Способ назначения имени модулю. Действует следующее правило:

1. Если директива NAME присутствует, то ее операнд становится именем модуля.
2. Если директива NAME отсутствует, то именем модуля становятся 6 первых символов в директиве TITLE.
3. Если директивы NAME и TITLE отсутствуют, то именем модуля становится имя файла.

Выбранное имя передается ассемблером в компоновщик.

Директива LABEL.

Директива позволяет переопределять атрибут определенного имени. Например:

LOO LABEL BYTE

NA DW 1234H

LOO указывает на первый байт переменной NA.

Директива RECORD.

Предназначена для определения двоичного набора в байте или слове.

Формат директивы: имя RECORD имя_поля:ширина[=выражение][,..] .

Например:

BIT RECORD B1:3,B2:10,B3:3 -определяет запись, состоящую из трех полей.

BIT1 RECORD BI1:10=1010110011B,BI2:6=111001B -определяет запись с инициализацией.

Для того чтобы отвести память под определенные записи далее в тексте программы по данному шаблону должно быть зарезервировано место для данных полей. Зарезервированному слову или байту присваивается имя: DEFB BIT1 <>. Причем в квадратных скобках могут быть переопределены или вновь определены поля. Далее в тексте программы переменная DEFB может быть использована в командах типа MOV BX,DEFB.

С директивой RECORD работают также директивы WIDTH и MASK. Директива WIDTH определяет длину в битах поля записи или всей записи: MOV AL,WIDTH BI2 - засылает в FL число 6. Директива MASK возвращает маску из единичных битовых значений, которые определяют битовые позиции, занимаемые данным полем. Ниже показано соответствие полей и масок к ним:

```
B1  1110000000000000
B2  0001111111111000
B3  0000000000000111
```

Наконец, имена полей также могут быть использованы в командах типа: MOV AL,BI1. При этом в AL будет загружено число, на которое нужно сдвинуть BI1, для того чтобы выровнять поле по правому краю. К примеру, команда MOVCL,B1 загрузит в CL число 13, а команда MOV CL,B3 - число 0.

Макроопределение состоит из заголовка: имя MACRO параметры, тела макроса и конца определения ENDM. Ниже приводятся два примера макроопределений.

```
;макроопределение вывод строки
;вызов: OUT_STRING ST , где ST указывает на строку
OUT_STRING MACRO STRING
      MOV DX,OFFSET STRING
      MOV AH,9
      NT 21H
ENDM

;макроопределение установка курсора
;вызов: LOCATE 12,20
LOCATE MACRO X,Y
      MOV BH,0
      MOV AH,2
```

```
        MOV  DH,Y
        MOV  DL,X
        INT  10H
ENDM
```

Макроопределения представляют собой шаблоны, которые вставляются в программу в том месте, где они вызываются. Использование макроопределений вместо процедур может увеличить быстродействие программы, но увеличивает ее объем. Макроопределение может быть внутри другого макроопределения, т.е. допускается вложенность макроопределений.

Часто для удобства создают библиотеки макроопределений, которые могут быть подключены при помощи директивы INCLUDE. Здесь могут встретиться следующие проблемы:

1. Часть макроопределений может не использоваться программой, но во время трансляции они будут загружаться в память и может появиться сообщение о нехватке памяти. Это устраняется директивой PURGE, при наличии которой макроопределения, указанные в ней, удаляются из текста. Формат использования этой директивы имеет вид: PURGE имя_макро1,имя_макро2..
2. При написании макроопределения может возникнуть потребность перехода на метку. Поскольку макроопределение может вызываться несколько раз, то обычная метка не годится. В пределах, однако, данного макроопределения можно ввести локальные метки при помощи директивы LOCAL.
3. Поскольку мы рассматриваем двухпроходной ассемблер, то возникает потребность избежать ситуации, когда файл, указанный в INCLUDE, будет загружаться два раза, используя память и дважды появляясь в листинге. Эта проблема устраняется использованием конструкции:

```
IF1
        INCLUDE   имя_файла
ENDIF
```

Существуют и другие условные конструкции, действующие на процесс ассемблирования и особенно удобные для использования внутри макроопределений. Общий вид их:

```
IFXX  условие
        .
        .
        ELSE
        .
        .
        ENDIF
```

Ниже мы перечисляем их:

IF выражение - если выражение равно нулю, то ассемблер обрабатывает выражение в условном блоке.

IFE выражение - если выражение не равно нулю, то ассемблер обрабатывает выражение в условном блоке.

IF1 - если осуществляется первый проход, то обрабатывается выражение в условном блоке.

IF2 - если осуществляется второй проход, то ассемблер обрабатывает выражение в условном блоке.

IFNDEF идентификатор - если идентификатор не определен в программе и не объявлен как EXTRN, то ассемблер обрабатывает операторы в условном блоке.

IFB <аргумент> - если аргумент пустой символ, то ассемблер обрабатывает операторы в условном блоке. Аргумент должен быть в угловых скобках.

IFNB <аргумент> - противоположное предыдущему.

IFIDN <ARG1>, <ARG2> - если строка первого аргумента идентична строке второго аргумента, то ассемблер обрабатывает операторы в условном блоке.

IFDIF <ARG1>, <ARG2> - противоположное предыдущему.

Директивы повторения.

REPT выражение - осуществляет повторение блока операторов до директивы ENDM в соответствии с выражением.

IRP N,<аргументы> - в угловых скобках содержится любое число правильных символов, строк, чисел. Использование директивы иллюстрирует следующий пример:

```
IRP N,<2,33,67,33,77>
      DB N
ENDM
```

Ассемблером будет сгенерирована последовательность: DB 2, DB 33, DB 67, DB 33, DB 77.

IRPC N, строка - ассемблер генерирует блок кода для каждого символа в строке.

Директива выхода EXITM.

Используется в макроопределениях.

Формат использования:

```
IFXX
.
.
EXITM
.
.
ENDIF
```

Как только ассемблер доходит до директивы EXITM, то переходит к командам за ENDM.

В последних версиях Макроассемблеров появились операторы проверки условий, которые весьма напоминают операторы языков высокого уровня. Вид этих операторов следующий:

```
.IF  усл.
      .
      .
ELSE
      .
      .
.ENDIF
```

Причем в условии могут содержаться регистры и переменные. Условия могут сопрягаться друг с другом при помощи логических связок. Скажем, следующий фрагмент

```
      cmp ax,0
      jnz _no
      add ax,bx
_no:
```

может быть представлен как

```
.IF ax=0
      add ax,bx
.ENDIF
```

Неправда ли, весьма удобное нововведение? Впрочем, я по-прежнему считаю, что для начинающего программировать на ассемблере эти операторы принесут скорее вред, чем пользу.

Директива **STRUC.**

Данная директива предназначена для объединения различных полей под одним именем. Формат ее таков:

```
имя_структуры STRUC
      ...
      определение  полей
      ...
имя_структуры  ENDS
```

Далее в программе должна стоять директива, выделяющая память для структуры указанного типа:

имя **имя_структуры** <>. В угловых скобках можно указать начальное значение полей по порядку. Например:

MSG MESSA <0,0,0,0,0,0,100,100>. Если значение параметров не указывается, то оно обнуляется.

Предположим, в программе записан следующий шаблон структуры:

```
NAMES   STRUC
        NAME1  DB  12  DUP(0)
        NAME2  DB  12  DUP(0)
NAMES   ENDS
```

Далее в программе следует записать директиву, чтобы выделить память для структуры:

```
NAME_F   NAMES   <>
```

После этого я могу обращаться к полям структуры следующим образом:

LEA DX,NAME_F.NAME2 - загрузить смещение поля NAME2 в DX и т.п.

Резервирование памяти (переменные).

DB - байт (1)
DW - слово (2)
DD - двойное слово (4)
DF - шесть байт (6)
DQ - восемь байт (8)
DT - десять байт (10)

Порядок следования сегментов. Директивы появились, начиная с версии 5.0.

.SEQ - сегменты следуют тому порядку, как они расположены в файле. Такой порядок, как известно, действует по умолчанию.

.ALPHA - сегменты следуют в алфавитном порядке.

.DOSSEG - расположение сегментов соответствует принятому в языках высокого уровня фирмы Microsoft. Появилась с версии Masm 6.O. Использование см. главу 24.

Приложение 4. О системном отладчике DEBUG.

Наше изложение будет относиться к базовой части утилиты DEBUG, которая существовала еще в старых версиях.

Данная утилита - один из самых простых отладчиков. Однако в этом ее достоинство. Кроме того, она не занимает много места в памяти и, следовательно, позволяет загружать в память даже очень большие программы.

Загрузить программу в среду отладчика можно непосредственно из отладчика (см. ниже) либо указав имя программы в командной строке вместе с необходимыми для нее параметрами. Загружать можно как COM-, так и EXE-программы.

Команды системного отладчика.

Каждая команда состоит из одной буквы и параметров. Отмена любой команды в любой момент производится нажатием клавиш CTRL C. Все числа по умолчанию считаются записанными в шестнадцатеричном формате.

A [адрес] - переход в режим ввода ассемблерных команд. Под адресом здесь и далее понимается одно- или двухкомпонентная величина. Например, возможны следующие команды: A 04BC:0200 A CS: 1000 A 300 - подразумевается CS:300 A - подразумевается CS: 100 для COM-программ и CS:0000 для EXE-программ. Выход из режима ввода ассемблерных команд происходит путем ввода пустой строки.

C диапазон адрес - сравнение областей памяти. Диапазон задается двумя адресами. Сообщение о несовпадающих адресах выдается в формате: адрес1 байт1 байт2 адрес2.

D [диапазон] - выдача дампа - содержимое памяти в заданном диапазоне. Если параметр не указан, то выдается 128 байт, начиная с адреса DS: 100.

E адрес - ввод байта. При этом выводится значение указанного байта и предлагается ввести новое значение.

F диапазон список - заполнение области. Данная команда заполняет область памяти. Если в список входит больше одного байта, то, соответственно, область заполняется двойками, тройками и т.п. Байты пишутся через пробел.

G [=адрес начала [адреса контрольных точек]] - команда выполнения программы. Можно указать до 10 контрольных точек. В контрольных точках происходит остановка выполнения. Для продолжения выполнения служит G без параметров.

I адрес порта - вывод из порта. Выводит значение, содержащееся в порте.

N имя файла -установка текущего имени файла. Если файл был загружен через командную строку, то данная команда назначает параметры для загруженного файла.

L - загружает файл, указанный через команду N. При этом в BX:CX будет содержаться длина файла в байтах. Загрузка осуществляется корректно как для COM-, так и для EXE-программ.

M диапазон адрес - перемещение данных из одной области в другую.

O адрес порта байт - заносит байт в указанный порт.

R [регистр] -вывод и ввод в регистры. Команда R выводит содержимое всех регистров. R AX выводит содержимое AX и предлагает ввести новое значение.

S диапазон список байт - поиск указанных байт в заданном диапазоне.

T [=адрес[величина]] - команда трассировки. Выполнение команды в пошаговом режиме. После каждого шага выдается значение регистров и следующая команда. Второй параметр задает количество команд, после которых происходит прерывание.

U [диапазон] - дизассемблирует указанную область памяти. Вместо диапазона можно указать адрес. Тогда дизассемблируется 32 байта.

W - запись программы на диск. Запись осуществляется только в формате COM. Т.о. если программа была загружена в формате EXE, то нам не удастся правильно записать ее на диск.

Q - выход в операционную систему.

Написание программ в среде отладчика.

Используя команду A, Вы можете написать программу прямо в отладчике. Затем определите ее имя при помощи команды N, укажите длину в регистрах BX:CX и программа готова. Ее можно записать на диск при помощи команды W. Весьма удобный способ написания небольших COM-программ.

Программы для отладчика.

Довольно интересно, что отладчик может выполнять ассемблерные программы, написанные в текстовом формате. Возможность эта обусловлена средствами перенаправления ввода-вывода в операционной системе MS DOS. Основная идея заключается в том, что при вводе с клавиатуры в среде отладчика как раз используется стандартный ввод, который можно осуществить и из файла. Например, команда DEBUG < A.DBG осуществит ввод информации в отладчик из файла. Заметим также, что концы строк в текстовом файле отмечены кодами возврата каретки (13,10), что позволяет при загрузке текстового файла автоматически выполнять каждую вводимую строку.

```
A
MOV AH, 2
MOV DL,7
INT 21
RET

G
Q
```

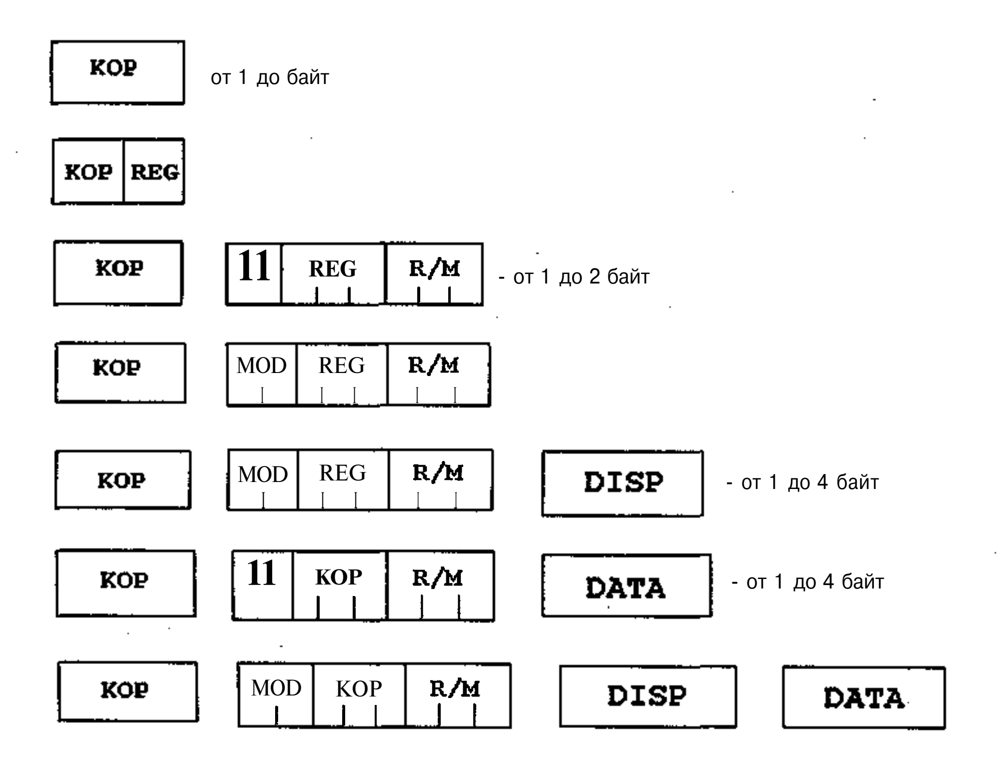
Выше представлена небольшая программа, производящая звуковой сигнал (код 7). Обращаю Ваше внимание на то, что после команды RET стоит пустая строка, имитирующая просто нажатие клавиши ENTER, что, в свою очередь, перыеводит отладчик из режима ввода команд ассемблера к командному режиму отладчика.

Приложение 5. Форматы машинных команд.

Здесь кратко представлены форматы команд микропроцессора Intel на двоичном уровне. Более подробно с ними можно познакомиться в [2, 4, 6]. Семь возможных форматов команд представлены ниже на рисунке.

Введены следующие обозначения:

KOP - код операции
REG - регистр
MOD - режим
R/M - регистр или память
DISP - смещение
DATA - непосредственные данные.



Ниже приводятся примеры на каждый из форматов.

```
1). NOP          90H
    90H - KOP

    HLT          F4H
    F4H - KOP

2). INC AX       40H
    01000B - KOP
    000B   - AX (REG)

    DEC AX       48H
    01001B - KOP
    000B   -AX (REG)

    INC BX       43H
    01000B - KOP
    011B   - BX (REG)

    DEC BX       4BH
    01001B - KOP.
    011B   - BX (REG)
```

Команды первой и второй группы однобайтные. Однако часть кода в командах второго типа отведена под поле REG, определяющее REG.

```
3). MOV AX,BX   89D8H
    89H - KOP
    11
    011 - BX (REG)
    000 - AX (R/M)

    ADD AX,CX   01C8H
    01H  - KOP
    11B
    001B - CX (REG)
    000B - AX  (R/M)

4) . MOV BX, [SI] 8B1CH
    8BH  - KOP
    00B  - MOD
    011B - BX  (REG)
    100B - [SI]  (R/M)
```

```
    MOV AX,[DI] 8B05H
    8BH  - KOP
    00B  - MOD
    000B - AX (REG)
    101B -  [DI]  (R/M)

    JMP  [SI]    FF24H
    FFH  - KOP
    00B  - MOD
    100B - REG
    100B -  (R/M)

5). ADD AX,[106H]   03060601H
    03H   - KOP
    00B   - MOD
    000B  - AX (REG)
    HOB  - R/M
   0601H   - DISP (106H)

6). SUB BX,43A4H   81EBA443H
    81H  - KOP
    11B
    101B - KOP
    011B - BX (R/M)
    A443H - DATA (43A4H)

7).MOV [12A7H],3456H   C706A7125634
   C7  - KOP
   00B - MOD
  000B - KOP
  HOB - R/M
 A712H - DISP
 5634H - DATA
```

После представленных схем и примеров читатель, я думаю, без труда сможет разобрать любую команду на отдельные поля.

Представленный материал не является необходимым при программировании на языке ассемблера. Транслятор делает все сам. Лишь при написании очень специфических программ, связанных с самомодификацией кода программы, эта информация может Вам пригодиться.

Приложение 6. Список векторов прерываний (кроме вызовов функций BIOS и DOS).

Данная информация взята автором из различных компьютерных источников. Более полную информацию о векторах прерываний можно получить в известном электронном справочнике INTER, где собран столь огромный объем информации, что приходится только удивляться. На устаревшую информацию, представленную в данном приложении, смотрите как на исторический экскурс.

INT 0 - деление на ноль.

Прерывание через вектор 0 генерируется процессором в ситуации, когда результат выполнения инструкции DIV или IDIV не помещается в регистр или ячейку памяти, а также при попытке деления на 0. BIOS устанавливает пустой обработчик прерывания О, возвращающий управление следующей инструкции процессора. При загрузке MS DOS устанавливается другой обработчик INT 0. Он выводит на экран сообщение "Деление на ноль" и возвращает управление MS DOS. Обычно MS DOS прекращает выполнение прикладной программы, вызвавшей это прерывание.

Замечание: в компьютерах на основе микропроцессоров Intel 80286 и выше обработчик INT 0, аналогичный обработчику MS DOS, устанавливается BIOS, поскольку эти микропроцессоры при прерывании 0 загружают в стек адрес инструкции DIV (IDIV), а не адрес следующей инструкции.

INT 1 - режим пошагового выполнения программы.

Прерывание через вектор 1 генерируется после выполнения любой инструкции в тех случаях, когда флаг TF процессора установлен в 1. Флаг TF сбрасывается любым прерыванием, так что обработчик пошагового режима не будет сам выполняться в этом режиме, но после выхода из него TF будет вновь установлен.

BIOS устанавливает пустой обработчик INT 1, так что установка флага TF приведет лишь к (значительному) замедлению выполнения программы. MS DOS не изменяет и не использует INT 1, но это прерывание используется в большинстве отладчиков, в частности, в отладчике DEBUG MS DOS.

INT 2 - немаскируемое прерывание.

Прерывание генерируется при выполнении некоторых зависящих от модели компьютера условий, обычно связанных с ошибками. Во многих моделях компьютеров немаскируемое прерывание генерируется при сбоях памяти. В некоторых моделях это прерывание используется для сигналов процессору от арифметического сопроцессора. Обработчик прерывания 2, устанавливаемый BIOS, зависит от модели компьютера. В старых советских компьютерах "Искра-1030", где немаскируемое прерывание связано с ошибками памяти, на экран выводилось сообщение "НП", и процессор вхо-

дил в бесконечный цикл. В других компьютерах сообщение может содержать адрес ячейки памяти, вызвавшей ошибку.

MS DOS не изменяет и не использует обработчик немаскируемого прерывания, однако выполняемые под управлением MS DOS программы, распознающие и использующие арифметический сопроцессор, могут устанавливать собственные обработчики INT 2 для его поддержки.

Прерывание 2 - единственное прерывание, которое нельзя замаскировать очисткой флага IF процессора (инструкцией CLI). Во многих компьютерах предусмотрен специальный порт (см. главу 26) для маскирования этого прерывания.

Замечания:

1. В PC jr немаскируемое прерывание присоединялось к прерыванию от клавиатуры.
2. В System/2, исключая модель 30, немаскируемое прерывание выдавало на экран один из четырех кодов ошибки:
   110 - сбой памяти на системной плате;
   111 - активна проверка канала ввода-вывода (предполагается наличие памяти в канале);
   112 - тайм-аут сторожа (Watchdog) - обнаружено прерывание от некоторого таймера, мешающее работе системы;
   113 - тайм-аут шины НДП - некоторое устройство пытается захватить шину данных больше, чем на 7.8 мксек.
3. В PC Convertible немаскируемое прерывание присоединялось к прерываниям от клавиатуры, дискетных устройств, часов и вызывалось проверкой канала ввода-вывода.
4. В системах с сопроцессором 8087 ошибка сопроцессора вызывает немаскируемое прерывание.
   В системах с сопроцессором 80287 или 80387 ошибка сопроцессора вызывает прерывание через линию IRQ 13; обработчик этого прерывания для совместимости вызывает прерывание 2.
5. Если пользователь хочет обрабатывать немаскируемые прерывания от сопроцессора, то он должен обрабатывать и другие ошибки. В частности, если при выполнении обработчика немаскируемого прерывания происходит ошибка немаскируемого прерывания, то управление должно быть передано системному обработчику INT 2.

INT 3 - точка прекращения.

Для вызова прерывания 3 предусмотрена специальная инструкция процессора INT 3, отличающаяся от других инструкций INT тем, что она имеет длину один байт. Это позволяет многим отладчикам (в частности, DEBUG) использовать прерывание 3 для установки точки прекращения программы. Отладчик заменяет байт инструкции в точке прекращения байтом инструкции INT 3 и восстанавливает его при достижении точки прекращения.

BIOS устанавливает пустой обработчик INT 3, который не заменяется и не используется MS DOS.

INT 4 - арифметическое переполнение.

Прерывание через вектор 4 вызывается специальной инструкцией INTO, если при ее выполнении установлен в 1 флаг OF процессора. Оно может быть также вызвано обычной инструкцией прерывания (INT 4) независимо от состояния OF.

BIOS устанавливает пустой обработчик INT 4, который не заменяется и не используется MS DOS. Многие программы, в частности, компиляторы и программы, ими генерируемые, устанавливают собственные обработчики INT 4, сигнализирующие об арифметическом переполнении.

INT 5 - печать экрана.

Прерывание 5 устанавливается BIOS ПП ЭВМ любой модели на выполнение функции печати экрана. При выполнении этого прерывания, которое может быть вызвано инструкцией INT 5 программы или нажатием специальной клавиши или комбинации клавиш на клавиатуре компьютера, содержимое экрана выводится на устройство печати LPT1.

Вызов INT 5 не требует и не возвращает никакой информации в регистрах процессора, но устанавливает код состояния печати экрана в байте памяти с адресом 0000:0500H:

OOH - функция не вызвана или завершена;

01H - функция выполняется;

OFFH - обнаружена ошибка при выполнении функции (обычно устройство печати выключено).

Функция печати экрана выполняется при разрешенных прерываниях, так что, вообще говоря, состояние экрана может быть изменено во время печати. Позиция курсора на экране сохраняется перед выполнением функции и восстанавливается при ее завершении.

Функция выполняется независимо от установленного режима дисплея (см. функцию 0 INT 1OH), но на печать выводится только содержимое текстового экрана. В графических режимах области экрана, не содержащие изображений символов, отображаются пробелами.

MS DOS не заменяет обработчик прерывания 5, но команда MS DOS GRAPHICS устанавливает резидентный обработчик, позволяющий копировать графический экран.

Печать экрана прекращается по ctrl+BREAK.

INT 8 - прерывание от системного таймера.

Через вектор 8 выполняется аппаратное прерывание (IRQ 0), активируемое системным таймером 18.2 раза в секунду (примерно через 55 мсек). BIOS устанавливает обработчик прерывания, который:

- инкрементирует счетчик прерываний от системного таймера;
- декрементирует счетчик времени до выключения двигателя дискетных устройств;
- генерирует прерывание 1CH.

Счетчик прерываний от системного таймера хранится в формате длинного целого (32 бита, младшее слово предшествует старшему) в ячейке памяти по адресу 0000:046CH. Когда содержимое счетчика достигнет значения, соответствующего 24 часам, то оно сбрасывается в 0, а в байте с адресом 0000:0470 устанавливается флаг этого события.

Счетчик времени до выключения двигателя дискетных устройств устанавливается в значение, обычно соответствующее двум секундам, после выполнения любой операции доступа к дискете. Каждое прерывание от системного таймера декрементирует счетчик до тех пор, пока он не сбросится в 0. В этот момент двигатель дискетных устройств будет выключен. Счетчик времени до выключения двигателя располагается в слове по адресу 0000:0440.

Прерывание 1CH резервируется для прикладных программ, желающих использовать системный таймер для своих целей.

MS DOS не заменяет обработчик прерывания 8.

INT 9 - прерывание от клавиатуры.

Контроллер клавиатуры генерирует запрос на прерывание 9 (IRQ 1) всякий раз, когда нажимается или отпускается какая-либо клавиша. Если какая-либо клавиша, исключая некоторые регистровые клавиши, остается нажатой более 0.5 сек, то контроллер выдает повторные прерывания как бы по новому ее нажатию каждые 0.1. сек. Контроллер содержит внутренний буфер, способный хранить информацию о нескольких (обычно о двадцати) клавиатурных действиях, при переполнении внутреннего буфера информация теряется.

BIOS устанавливает вектор 9 на адрес обработчика клавиатурных прерываний, который принимает и удаляет из внутреннего буфера клавиатуры код нажатия или отпускания клавиши (скан-код) и преобразует его в код символа по следующим правилам:

1. Если нажата клавиша, которой соответствует какой-либо символ кода, то скан-код клавиши и код символа размещаются в буфере. Буфер клавиатуры длиной 32 байта располагается в области памяти, начиная с адреса 0000:041 EH, и представляет собой кольцевой буфер. Указатель конца буфера (т.е. указатель свободной памяти в буфере) располагается в слове по адресу 0000:041CH. После приема знака указатель буфера увеличивается на 2, и если он начинает указывать за пределы буфера, то устанавливается на начало. Вырабатываемый код символа зависит от состояния регистров клавиатуры.

2. Если нажата или отпущена регистровая клавиша, то в буфер не заносится никакой информации, а изменяются байты состояния регистров клавиатуры 0000:0417H и 0000:0418H. Из регистровых клавиш высший приоритет имеет клавиша Alt, затем Ctl и, наконец, клавиша перехода на верхний регистр Shift. Соответствующие состояния устанавливаются только на время, пока нажата регистровая клавиша, и сбрасываются при ее отпускании. Остальные регистровые клавиши - CapsLock, Num Lock, Scroll Lock - действуют как триггеры: каждое нажатие переключает соответствующее состояние. Регистр Shift действует только на алфавитные клавиши, набор которых зависит от текущей установки алфавита (состояния РУС/ЛАТ), инвертируя для них смысл нажатой клавиши перехода на верхний регистр. Регистр Num Lock аналогично действу-

ет на клавиши цифровой клавиатуры, позволяя вырабатывать цифровые коды без перехода на верхний регистр. Обычно в таком случае вырабатываются расширенные коды управления курсором.

3. Последовательность нажатий цифровых клавиш в состоянии Alt рассматривается как поступление десятичных цифр кода одного символа и сохраняется в драйвере. При отпускании клавиши Alt остаток от деления на 256 введенного числа помещается в буфер вместе со скан-кодом 0.

4. Нажатие клавиши Del в состоянии Alt+Ctrl (т.е. при одновременно нажатых клавишах Alt и Ctrl) вызывает системный сброс: флаг сброса - слово по адресу 0000:0472 - устанавливается в 1234H, и управление передается на тесты самопроверки. Поскольку флаг клавиатурного сброса установлен, то при выполнении тестов обходятся тесты ОЗУ.

5. Нажатие клавиши Num lock в состоянии (т.е. вместе с) Ctrl приостанавливает выполнение любой программы до нажатия какой-либо клавиши.

6. Нажатие клавиши Print Screen на верхнем регистре вызывает прерывание 5, т.е. вывод на печать копии экрана.

7. Нажатие клавиши Scroll Lock в состоянии Ctrl вызывает прерывание 1BH (Break).

8. Отпускание любой клавиши, кроме некоторых регистровых, игнорируется.

Для AT, PC Convertible, System/2 и некоторых моделей XT (BIOS 1986 г.) перед выполнением системного сброса вызывалась функция (AH)=85H прерывания 15H, чтобы информировать систему о подготовленном сбросе. Кроме того, клавиатурное прерывание может вызывать прерывание 15H с (AX) = 9102H, чтобы сообщить системе о наличии знака в буфере.

В некоторых моделях компьютеров (включая перечисленные выше) BIOS обеспечивает возможность преобразования скан-кодов клавиш перед размещением их в буфере. Для этого после приема каждого знака от клавиатуры вызывается функция (AH) = 4FH прерывания 15H, позволяющая перекодировать байт, принятый от клавиатуры (из порта 60H).

INT 19H - прерывание начальной загрузки.

Прерывание 19H вызывает чтение с дискового устройства первого сектора дорожки 0 на стороне 0. Сектор считывается в ОЗУ с адреса 7C00:0, и байт по этому адресу получает управление. Прежде всего выполняется попытка считать сектор с первого дискетного устройства (INT 13H с (DL) = 0), при неудаче - с первого жесткого диска (INT 13H с (DL) = 80H). Если и эта попытка оказывается неудачной, то выполняется прерывание INT 18, вызывающее в некоторых компьютерах кассетный Бейсик.

INT 1BH - прерывание прекращения.

Прерывание 1BH вызывается при нажатии комбинации клавиш Ctrl-Break (см. главу 9). BIOS инициирует обработчик этого прерывания, помещающий в буфер клавиатуры специальный код (AH) = 03, (AL) = 0. MS DOS устанавливает свой обработчик прерывания 1BH, прекращающий выполнение программы, вызвавшей это прерывание. Пользователь может установить собственный обработчик (некоторые утилиты MS DOS делают это).

INT 1CH - обработчик пользователя прерываний от таймера.

Прерывание 1CH вызывается из обработчика прерываний от системного таймера (INT 8) при каждом прерывании. BIOS устанавливает пустой обработчик прерывания 1CH; пользователь может заменять его для своих целей.

INT 1EH - указатель таблицы параметров дискеты.

BIOS устанавливает вектор 1EH на адрес в ПЗУ таблицы параметров дискетного устройства (см. главу 14).

INT 40H - функции дискетных устройств.

В компьютерах, снабженных жесткими дисками, драйвер BIOS жестких дисков переустанавливает вектор прерывания 13H так, чтобы поддерживать через этот вектор запросы к диску любого типа. Адрес обработчика запросов к дискете сохраняется в векторе 40H. Не рекомендуется ссылаться к этому вектору, за исключением случая, когда нужно получить адрес обработчика дискетных функций в ПЗУ BIOS.

INT 41H - указатель таблицы параметров жестких дисков.

В большинстве компьютеров таблицы параметров жестких дисков расположены в ПЗУ. Это сделано для того, чтобы можно было загружать операционную систему с жесткого диска любого типа. Типы установленных дисков обычно отображаются переключателями конфигурации, но некоторые адаптеры могут получать тип от устройства. Во всех случаях после инициализации BIOS вектор прерывания 41H устанавливается на таблицу параметров жесткого диска (если в системе есть жесткие диски). В большинстве компьютеров вектор указывает на начало таблицы в ПЗУ, а конкретный набор параметров выбирается на основании переключателей конфигурации, но в некоторых компьютерах (IBM AT, System/2) этот вектор устанавливается на таблицу параметров диска, фактически подключенного как жесткий диск 0. В этих компьютерах вектор 46H указывает на таблицу параметров жесткого диска 1, если он подключен.

Параметры жесткого диска представляются структурой, приведенной в следующей таблице.

адрес	длина	параметр
0	слово	максимальный номер цилиндра
2	байт	максимальный номер головки
3	слово	цилиндр начала уменьшения тока записи
5	слово	цилиндр начала предкомпенсации записи
7	байт	наибольшая длина пакета исправимой ошибки чтения

8	байт	байт управления: бит 7 =1 - запрет повторов поиска; бит 6 =1 - запрет повторов при ошибках чтения; бит 5 =1 - определен список дефектов; бит 4 - резерв; бит 3 = 1 - больше 8 головок; биты 2 - 0 - номер устройства
9	байт	стандартный тайм-аут
10	байт	тайм-аут для разметки
11	байт	тайм-аут для проверки
12	слово	Landing Zone
14	байт	число секторов на дорожке
15	байт	резерв

Параметры жестких дисков

Замечание. Не все параметры являются обязательными для контроллера каждого типа. В частности:

- может не быть цепей уменьшения тока записи;
- BIOS может не пользоваться значениями тайм-аутов;
- установленная длина исправимого пакета ошибок может не приниматься во внимание;
- любой из битов 6 и 7 байта управления может запрещать все повторы;
- список дефектных секторов, бит 3 байта управления и Landing Zone могут не поддерживаться;
- число секторов на дорожке не нужно для контроллеров, поддерживающих только фиксированный формат дорожки.

INT 70H - прерывание от часов реального времени.

В современных моделях компьютеров обработчик прерывания 70H управляет периодическими и разовыми прерываниями от часов реального времени.

Если пользователь определил какое-либо событие (см. Приложение 9), то активируются периодические прерывания от часов реального времени с частотой 1024 прерывания в секунду и устанавливается начальное значение счетчика. Каждое прерывание декрементирует счетчик, и, когда он сбросится в 0, будет установлен флаг события. Пользователь может проверять наступление события, анализируя флаг. Для функции (AH) = 86H INT 15H соответствующим флагом является бит 7 байта с абсолютным адресом 04A0H.

Установка пользователем сигнала тревоги активирует вызов прерывания 4AH из данного прерывания.

Приложение 7. Функции MS DOS.

Функции DOS представляют собой интерфейс, предоставляемый операционной системой программисту. Анализ функций показывает, что большая их часть отвечает за обслуживание файловой системы. Вывод на экран и обслуживание клавиатуры представлено на весьма примитивном уровне. Обслуживание мыши вообще отсутствует. Таким образом, система функций DOS заставляет программистов самим писать процедуры обработки внешних устройств. Это препятствует созданию стандартов в программировании, но стимулирует развитие творчества программистов. Кроме того, операционная система MS DOS является неинтерабильной, т.е. не позволяет осуществлять повторный запуск своих функций (точнее, осуществлять запуск функции из функции), что является серьезной помехой для создания резидентных программ (см. главу 12).

Вызов функций DOS:

```
MOV AH,NUM
{готовим другие регистры}
INT 21H
```

NUM - номер функции DOS, если функция выполнена с ошибкой, то устанавливается флаг переноса, а в AX заносится код ошибки. Иногда функция имеет подфункции. Номер подфункции обычно заносится в регистр AL.

Список функций DOS.

Функция 0.

Вход: CS - сегмент PSP программы.

Выход из программы. Аналогично INT 20H. Переустанавливает векторы 22H, 23H, 24H. Сбрасывает буфера, но, если длина файла менялась, файл предварительно следует закрыть. Удобнее пользоваться функцией 4CH.

Функция 1.

Вход:

Вводит символ со стандартного устройства ввода (обычно клавиатура) и выводит на стандартное устройство вывода (обычно экран). Допустимо перенаправление ввода. Для чтения расширенного кода требуется повторное чтение (первое чтение дает нуль). Реагирует на Ctrl-Break и Ctrl-C.

Функция 2.

Вход: DL - код символа.

Вывод символ на стандартное устройство вывода. Реагирует на Ctrl-Break и Ctrl-C.

Функция 3.

Вход:

Читает символ в AL из стандартного последовательного порта (COM1).

Функция 4.

Вход: DL - код символа.

Посылает символ в стандартный последовательный порт.

Функция 5.

Вход: DL - код символа.

Посылает символ на стандартное печатающее устройство LPT1:.

Функция 6.

Вводит или выводит символ со стандартных устройств.

Вывод: DL-0-FEH

Выводит символ на стандартное устройство.

Ввод: DL - FFH

В AL - возвращает код введенного символа, если символа нет возводится флаг нуля Z. Расширенный код вводится с повтором.

Функция 7.

Нефильтрованный ввод со стандартного устройства. Вводит символ без его отображения. Если символа нет, то ждет ввода.

Не реагирует на Ctrl-Break. В А1 - возвращает код введенного символа.

Функция 8.

Аналогична предыдущей функции, но реагирует на Ctrl-Break.

Функция 9.

Вывод строки.

Вход: DS:DX - адрес строки, в конце строки должно стоять '$'.

Функция АН.

Буферизованный ввод с клавиатуры.

Вход: DS:DX - адрес буфера. Структура буфера:

вначале байт максимальной длины строки, затем резервный байт, далее место для ввода текста. Ввод заканчивается либо по достижению максимального значения строки либо по нажатию возврата каретки (13).

Функция ВН.

Проверка состояния ввода. Если символ не ждет, то AL=0, если ждет, то AL=FFH.

Функция СН.

Очищает кольцевой буфер клавиатуры и активизирует функцию ввода.

Вход: AL - номер требуемой функции ввода: 01, 07, 08, ОАН. Для функции ОАН - DS:DX - адрес буфера. При возврате байт входных данных.

Функция ОЕН.

Назначает текущий диск и возвращает число логических дисководов в системе.

Вход: AL - код дисковода (0=А, 1=В и т.д.)

При возврате AL - число логических дисководов.

Функции 0FH-17H - работа с файлами через FCB. Устаревшие функции, сохраняемые ради совместимости. В настоящее время не используются, поэтому и мы их опускаем.

Функция 19H.

Возвращает код текущего диска.

Код диска возвращается в AL - O -A, 1 -B и т.д.

Функция 1AH.

Позволяет определить адрес области обмена с диском (DTA) для последующих операций с блоками управления файлами.

В DS:DX - адрес DTA.

Функция 1BH.

Возвращает характеристики текущего диска.

Выход:

AL - количество секторов в кластере,

CX - количество байтов в секторе,

DX - общее количество кластеров на диске,

DS:BX -> байт-описатель:

FFH- дискета 320K,

FEH- дискета 160K,

FDH- дискета 360K,

FCH- дискета 180,

F9H- дискета 1,2Мб,

F8H- жесткий диск,

F0H- другие носители.

Функции 1CH.

Возвращает характеристики заданного носителя.

Вход:

AL - номер носителя.

Выход: как в 1BH.

Функция 1FH.

Позволяет получить детальную информацию о параметрах текущего диска.

Выход:

AL - 0 -успешное завершение,

DS:BX - блок параметров: информация, содержащаяся в секторе загрузки.

Функции 21H-24H используют FCB, поэтому устарели и сохраняются только для совместимости.

Функция 25H.

Установка вектора прерывания.

Вход:

AL - номер вектора,

DS:DX - адрес программы обработки прерывания.

Функция 26H.

Создание префикса программного сегмента.
Вход:
DX - сегментный адрес для нового PSP. По этому адресу копируется текущий PSP.

Функции 27H-29H.

Используют FCB.

Функция 2AH.

Получение системной даты.
Выход:
CX - год,
DH - месяц,
DL - день,
AL - день недели (0 - воскресенье и т.д.).

Функция 2BH.

Установка системной даты.
Вход:
CX - год,
DH - месяц,
DL - день.
Выход:
AL=0 - успешное выполнение,
FFH - недопустимая дата.

Функция 2CH.

Получение времени.
Выход:
CH - часы,
CL - минуты,
DH - секунды,
DL - сотые доли секунды.

Функция 2DH.

Установка системного времени.
Вход:
CH - часы,
CL - минуты,
DH - секунды,
DL - сотые доли секунды.
Выход:
AL=0 - успешное выполнение,
FFH - недопустимое время.

Функция 2EH.

Установка флага проверки.
Вход:
AL=0 - установить флаг проверки,
AL= 1 - сбросить флаг проверки.

Функция 2FH.

Получение адреса текущей области обмена с диском (DTA).
Выход:
ES:DX - адрес DTA.

Функция 3OH.

Получение версии DOS.
Вход:
AL=0 - в BH вернуть номер OEM,
AL=1 - в BH вернуть флаг версии,
Выход:
AL - номер основной версии,
AH - номер подверсии,
BH - номер OEM:
00h - IBM,
16h - DEC,
99h - архитектура STARLITE,
FFH - Phoenix
BH - флаг версии:
08h - DOS находится в ПЗУ,
ЮН - DOS находится в области старшей памяти.

Функция 31h.

Завершение программы и сохранение ее резидентной.
Вход:
AL - код возврата,
DX - объем резервируемой памяти в параграфах.

Функция 32H.

Получение адреса блока параметров заданного диска.
Вход:
DL - номер диска.
Выход:
DS:DX - блок параметров.
При ошибке AL=0FFH.

Функция 33Н.

Получение или установка состояния Break (Ctrl Break, Ctrl C).
Вход:
AL=0 - получить состояние Break,
Al=1 - установить состояние Break,
DL=0 - состояние Break выключено, проверка только для функций 1-0Ch,
DL=1 - состояние Break включено.
Выход:
DL - текущее состояние Break, если при вызове AL=0.
Вход:
AL=2 - получение и установка состояния Break.
DL=0 - состояние Break выключено,
1 - cocnjzybt Break включено.
Выход:
DL - прошлое состояние Break.
Вход:
DL=5 - получить дисковод загрузки.
Выход:
DL - дисковод (1 - A, 2 - B и т.д.)

Функция 34Н.

Получение адреса флага занятости.
Выход:
ES:BX - однобайтовый флаг.

Функция 35Н.

Получение вектора прерывания.
Вход:
AL - номер вектора.
Выход:
ES:BX - значение вектора, т.е. адрес программы обработки прерывания.

Функция 36Н.

Получение объема свободного пространства на диске.
Вход:
DL - номер дисковода (0 - текущий, 1 - A: и т.д.).
Выход:
AX - число секторов на кластер,
BX - число свободных секторов,
CX - размер сектора в байтах,
DX - полное число кластеров на диске.
При ошибке AX=FFFFH.

Функция 37Н.

Смена символа разделителя в командной строке. По-видимому, с версии 6.0 DOS отсутствует.

Функция 38Н.

Получение/установка информации по стране.

Функция 39Н.

Создает каталог в конце указанного пути.
Вход:
DS:DX - путь в виде строки ASCII, в конце код 0.
Выход:
Если ошибка, то АХ - код ошибки.

Функция ЗАН.

Удаление указанного каталога.
Вход:
DS:DX - путь в виде строки ASCII, в конце код 0.
Выход:
Если ошибка, то АХ - код ошибки.

Функция ЗВН.

Смена текущего каталога.
Вход:
DS:DX - путь в виде строки ASCII, в конце код 0.
Выход:
Если ошибка, то АХ - код ошибки.

Функция ЗСН.

Создание или усекновение файла. Файл либо создается, либо, если он есть, усекается до нулевой длины. В любом случае он открывается.
Вход:
СХ - атрибут файла.
DS:DX - адрес спецификации файла в виде строки ASCII,
в конце - код нуля.
Выход:
АХ - дескриптор.

Функция 3DH.

Открыть файл. Если файла нет, то возводится флаг ошибки.
Вход:
AL - режимы доступа. Если к режиму добавлено 80Н, то дескриптор наследуется дочерним процессом.
DS:DX - адрес спецификации файла в виде строки ASCII, в конце - код нуля.
Выход:
АХ - дескриптор.

Функция ЗЕН.

Закрыть файл.
Вход:
ВХ - дескриптор.

Функция 3FH.

Чтение из файла или устройства.
Вход:
BX - дескриптор.
CX - число байт.
DS:DX - адрес буфера.
Выход:
AX - число считанных байт (или код ошибки).

Функция 40H.

Запись в файл или устройство.
Вход:
BX - дескриптор.
CX - число байт. Если =0, то длина файла усекается до положения указателя.
DS:DX - адрес буфера.
Выход:
AX - число переданных байт (или код ошибки).

Функция 41H.

Удаление файла.
Вход:
DS:DX - спецификация файла.

Функция 42H.

Установка указателя в файле.
Вход:
AL - режим установки указателя.
0 - абсолютное смещение от начала файла,
1 - знаковое смещение от текущего положения указателя,
2 - знаковое смещение от конца файла.
BX - дескриптор.
CX - старшая часть смещения.
DX - младшая часть смещения.
Выход:
DX - старшая часть возвращенного указателя.
AX - младшая часть возвращенного указателя.

Функция 43H.

Получение или установка атрибута файла.
Вход:
AL - 0 - для получения атрибута.
- 1 - для установки атрибута.
CX - атрибут.
DS:DX - адрес спецификации файла или каталога.
Выход:
CX - возвращаемый атрибут.

Функция 44H. IOCTL - Input-Output Control.

Взаимодействие с устройствами и получение информации о файлах. Номер подфункции помещается в регистр AL.

Подфункция 00. Запросить флаги информации об устройстве.

Вход:

BX - дескриптор файла или устройства,

Выход:

DX - информация об устройстве - 15 бит:

0 - 1 - консольное входное устройство,

1 - 1 - консольное выходное устройство,

2 - 1 - нулевое устройство (NUL),

3 - 1 - часы.

5 - режим (0 - ASCII, 1 - двоичный),

6 - 1 - нет конца файла, 0 - конец файла при вводе,

7 - 1 - устройство, 0 - файл

14 - 1 - строки IOCTL приняты

O - строки IOCTL нельзя обработать.

номер устройства: 0 - A, 1 - B и т.д.)

Подфункция 01. Установить флаги информации об устройстве.

Вход:

BX - дескриптор файла или устройства,

DX - информация об устройстве (DH=0).

Выход:

DX - информация об устройстве.

Подфункция 02-03. Запросить флаги информации об устройстве.

Читать (2), писать (3) строку на символьное устройство.

Вход:

DS:DX - адрес буфера чтения или записи,

CX - число передаваемых байт,

BX - дескриптор устройства.

Выход:

Подфункция 04-05. Читать (04) или писать (05) строку IOCTL на блочное устройство.

Вход:

DS:DX - адрес буфера чтения или записи,

CX - число передаваемых байт,

BL - номер диска (0 - текущий, 1 - A и т.д.)

Выход:

AX - действительное число переданных байт, если не было ошибки.

Подфункция 06-07. Дать статус ввода (06) или вывода (07).

Вход:

BX - дескриптор.

Выход:

AL = ffH - не конец файла, 0 - конец для дисковых описателей.

AL = FFH - готово, 0 - не готово для устройств.

Подфункция 08. Использует ли блочное устройство съемный носитель.

Вход:

BL - номер диска (0 - текущий, 1 - А и т.д.)

Выход:

АХ - 0 - съемный носитель, 1 - несъемный носитель (твердый диск или электронный диск)

Подфункция 09. Является ли устройство съемным в сети.

Вход:

BL - номер диска (0 - текущий, 1 - А и т.д.)

Выход:

DX - атрибут устройства:

0 - 1 - стандартное входное устройство,

1 - 1 - стандартное выходное устройство,

2 - 1 - стандартное устройство NUL,

3 - 1 - часы,

6 - 1 - поддерживает логические устройства,

11 - 1 - поддерживает съемные зависящие от носителей устройства,

12 - 1 - сетевое устройство (возможно, CD-rom),

13 - 1 - не IBM-блочные устройства,

14 - 1 - поддерживает IOCTL,

15 - 1 - символьное устройство, 0 - блочное устройство.

Подфункция ОАН. Принадлежит ли дескриптор файла локальному, или удаленному устройству в сети.

Вход:

ВХ - дескриптор файла.

Выход:

DX - атрибут устройства, если 15 бит = 1, то устройство удаленное (т.е. сетевое).

Подфункция 0BH. Контроль повторений при разделении и блокировании файлов.

Вход:

DX - число попыток перед вызовом критической ошибки,

СХ - счетчик цикла между попытками.

Выход:

Подфункция Ос. Поддержка переключений кодовых таблиц.

Вход:

ВХ - описатель открытого устройства,

СН - тип устройства:

0 - неизвестен,

1 - COMп - последовательное устройство,

3 - CON - консоль,

5 - параллельный принтер.

CL - код действия.

4ch - начало подготовки кодовой страницы,

4dh - конец подготовки кодовой страницы,

4ah - выбрать подготовленную кодовую страницу,

6ah - получить текущую активную кодовую страницу,

6bh - получить список подготовленных кодовых страниц

DS:DX - адрес пакета данных,

Выход:

Для подготовки кодовой страницы выдать вначале команду 4ch, а затем сделать несколько вызовов подфункции 3.

Подфункция 0dh. Вызов одной из функций управления.

Вход:

CL - код действия.

40h - установить параметры устройства,

60h - дать параметры устройства,

41h - писать дорожку логического устройства,

61h - читать дорожку логического устройства,

42h - форматировать дорожку с верификацией,

62h - верифицировать дорожку логического устройства.

DS:DX - адрес пакета данных.

Выход:

Подфункция Oeh. Выяснить, назначил ли драйвер устройства несколько логических устройств одному логическому устройству.

Вход:

BL - номер диска (0 - текущий диск, 1 - А и т.д.)

Выход:

Если не было ошибки то:

AL - 0 - назначен один диск, 1 - А и т.д.

Подфункция 0Fh. Сообщить драйверу блочного устройства номер диска для обработки.

Вход:

BL - номер диска (0 - текущий и т.д.)

Выход:

Если не было ошибки, то:

AL - 0 - назначен один диск, 1 - А и т.д.

Подфункция необходима для обхода сообщения "Insert diskete for drive ..."

Пакеты для функции 0ch IOCTL.

Функции 4AH, 4DH, 6AH.

```
DW ?  ;длина пакета
DW ?  ;код страницы
```

Функция 4CH.

```
DW ? ;флаги
DW ?  ;длина остатка пакета в байтах
DW ?  ;число последующих кодовых страниц
DW 15BH ;первая кодовая страница, например, США
DW ?  ;вторая кодовая страница
:
```

Функция 6BH.

```
Пакет возвращается в DS:DX.
DW ? ;длина списка в байтах
DW ? ;счетчик аппаратных кодовых страниц
DW ? ;аппаратная кодовая страница 1
DW ? ;аппаратная кодовая страница 2
.
.
DW ? ;счетчик подготовленных кодовых страниц
DW ? ;подготовленная кодовая страница 1
DW ? ;подготовленная кодовая страница 2
.
.
```

Пакеты для функции 0bh IOCTL.

Функции 40H,60H.

```
DB ?  ;специальные функции: для функции 60h определен
      ;только бит 0, 0=1 - извлечь BPB в формате запроса
      /устройства,   0=0 - извлечь умалчиваемый BPB
      /для функции 40H работают 3 бита:
      ;0 - 1 вернуть DeviceBPB в формате запроса устройства
      /О - 0 - использовать DeviceBPB из этого пакета
      ;1 - 1 - игнорировать все поля в пакете, кроме схемы
      ;дорожки
      ; 2 - 1 - все секторы на дорожке имеют один размер
DB ?  /тип устройства (возвращает драйвер):
      /О - 320/360 5.25 дискета
      /1 - 1.2М 5.25 дискета
      /2 - 720К 5.25 дискета
      /3 - 8-дюймовая одинарной плотности
      ; 4 - 8-дюймовая двойной плотности
      ;5 - фиксированный диск
      ;6 - ленточное устройство
      ; 7 - прочие устройства.
DW ?  /атрибут устройства (возвращает драйвер)
      ;определено два бита - 0 - 1 - носитель съемный, О
      ; - несъемный
      ;1 - 1 смена дискет поддерживается, 0 - не поддерживается
DW ? /число цилиндров (возвращает драйвер) - максимальное
     /число цилиндров, поддерживаемое физическим устройством
DB ?  /тип носителя - задаёт тип носителя для типа устройства,
      ;поддерживающего разные носители 1.2М устр:
      /О - 1.2М, 1 - 36OК.
DB 31 dup(?) /DeviceBPB. Блок BPB + 12байт
```

```
;---------------
   DeviceBPB:
   Блок BPB (13 байт, см. глава 14, Рис. 14.4)
   DW ? ;секторов на дорожку
   DW ? ;головок на устройстве
   DD ? ;секторов, не используемых драйвером
   DB 10 dup(?) ;резерв
;----------------
   DB ?  ;схема дорожки (переменной длины)
;----------------
   Схема дорожки:
   DW ? ;всего секторов на дорожке
   DD ? ;номер и размер сектора (например, DW 1,200H)
   .
   .
;----------------
```

Функции 41H,61H.

```
DB  ? ;специальные функции (всегда 0)
DW  ? ;головка чтения/записи
DW  ? ;дорожка чтения/записи
DW  ? ;начальный сектор (от 0)
DW  ? ;счетчик секторов (от 0)
DD  ? ;адрес буфера пользователя
```

Функции 42H,62H.

```
DB  ? ;специальные функции (только бит 0)
      ;=0 необычная структура поддерживается
      ;=1 необычная структура неприемлема
DW  ? ;головка
DW  ? ;дорожка (цилиндр)
```

Продолжение по функциям DOS.

Функция 45H.

Создает новый дескриптор файла, который связан с заданным файлом или устройством через тот же элемент системной таблицы файлов.

Вход:
BX - дескриптор файла
Выход:
AX - новый дескриптор.

Функция 46H.

Принудительно объявляет указанный дескриптор дубликатом заданного. Если дескриптор в BX был открыт, то он закрывается.

Вход:
BX - дескриптор файла,
CX - дескриптор, который должен стать дубликатом первого.

Функция 47H.

Возвращает полный путь к текущему каталогу.
Вход:
DL - дисковод (0 - текущий, 1 - А и т.д.)
DS:SI - адрес буфера (64 байта).
Выход:
В буфере содержится путь.

Функция 48H.

Выделение блока памяти.
Вход:
BX - требуемое число параграфов.
Выход:
AX - сегментный адрес блока.

Если памяти не хватает, то, как обычно, возводится флаг переноса, а в BX заносится размер наибольшего доступного блока.

Функция 49H.

Освобождение блока памяти.
Вход:
ES - сегментный адрес освобождаемого блока.

Функция 4AH.

Изменяет размер выделенного блока памяти.
Вход:
ES - сегментный адрес блока,
BX - требуемый размер блока в параграфах.
Выход:
В случае ошибки - BX содержит размер наибольшего доступного блока в параграфах.

Функция 4BH.

Запуск программы (дочернего процесса). Более подробно работа с этой функцией разбирается в главу 11.

Вход:
AL = 0 - загрузить и выполнить программу,
= 1 - загрузить и не выполнять программу,
= 3 - загрузить оверлей.
ES:BX - адрес блока параметров (см. глава 11)
DS:DX - путь к запускаемой программе.

Функция 4CH.

Завершение процесса.
Вход:
AL - код возврата.

Функция 4DH.

Получение кода возврата и типа завершения дочернего процесса.
Выход:
AH = 0 - нормальное завершение
1 - завершение через Ctrl+Break
2 - завершение через драйвер критической ошибки
3 - завершение через int 21H или 31h функцию.
AL = код возврата.

Функция 4EH.

Нахождение первого файла согласно указанным параметрам.
Вход:
CX - атрибут файла
DS:DX - адрес строки, указывающей путь к файлу.
Выход:
полное имя файла заносится в DTA.

Функция 4FH.

Осуществляет поиск следующего файла.

Выполняется после функции 4EH и далее до тех пор, пока не появится флаг переноса. AX в конце будет содержать код - файлов больше нет.

Ниже на рисунке представлена структура DTA.

см.	длина	что находится	пояснение
40	21	зарезервировано	используется функцией 4FH
+15H	1	атр	атрибут найденного файла
+16H	2	время	время создания/модификация
+18H	2	дата	дата создания/модификация
+laH	4	размер файла	DWORD
+leH	13	имя файла	имя,точка,расширение,нуль

Функция 50H.

Устанавливает текущий PSP. Документирована, начиная с версии 5.0.
Вход:
BX - сегментный адрес PSP, объявляемый текущим.

Функция 51H.

Получить PSP текущего процесса.
Выход:
BX - PSP текущего процесса.

Функция 52H.

Получить адрес списка списков.
Выход:
ES:BX - адрес списка списков.

Функция 54H.

Получить флаг "verify".
Выход:
AL - 0 выключен
1 включен.

Функция 56H.

Переименовать файл.
Вход:
DS:DX - адрес спецификации файла.
ES:DI - адрес новой спецификации файла.

Функция 57H.

Подфункция 0.
Получение даты и времени создания или модификации файла.
Вход:
BX - дескриптор файла.
AL=0.
Выход:
CX - время (0-4 сек., 5-Ah мин., Bh-Fh час.
DX - дата (0-4 ден., 5-8 месяц, 9-Fh - год относит. 1980).
Подфункция 1.
Модификация даты-времени создания файла.
BX - дескриптор.
AL=1.
CX,DX - время и дата (см. подф. 0).

Функция 59H.

Позволяет получить детальную информацию об ошибке. Обычно используют в обработчике критической ошибки.

Вход:
BX = 0
Выход:
AX - расширенный код ошибки,
BH - класс ошибки,
BL - рекомендуемое действие,
CH - местоположение ошибки.
Кроме указанных регистров, портятся также регистры: DX, SI, DI, BP, DS, ES.

Функция 5AH.

Создает временный файл. Имя файла определяет система.
Вход:
CX - атрибут файла
DS:DX - полное имя каталога (строка)
Выход:
DS:DX - полное имя файла
AX - код ошибки

Функция 5BH.

Создание нового файла.
Отличается от 3CH только тем, что если указанный файл уже существует, то функция завершается ошибкой.

Функция 5DH.

Вход:
AL = 6
Адрес области текущих данных. Здесь хранятся системные переменные и расположены системные стеки.
Выход:
DS:SI - адрес области
CX - размер в байтах части области, которая должна сохраняться при переходе на другой процесс, если прерывается функция DOS
DX - размер в байтах части области, которая должна сохраняться при переходе на другой процесс во всех случаях
Вход:
AL = Oah
DS:DX - 3-словный список параметров, составляющих расширенную информацию об ошибке.
Выход:
Восстанавливает в области текущих данных расширенную информацию об ошибке. Эта информация предварительно может быть получена с помощью функции 59H.

Функция 62H.

Получение PSP текущего процесса.
Выход:
BX - сегментный адрес PSP.

Функция 67H.

Установить число дескрипторов файлов.
Вход:
BX - требуемое число дескрипторов.
Выход:
В PSP будет записан адрес новой таблицы дескрипторов.
Для успешной работы этой функции требуется достаточно свободной памяти (см. главу 8).

Функция 68H.

Сбрасывает буфера для данного файла на диск, не закрывая файл.
Вход:
BX - дескриптор

Функция 69H.

Установка метки тома и серийного номера.
Вход:
BL - дисковод (0 - текущий, 1 -А и т.д.)
AL - 0 - получит метку тома
1 - установить метку тома
DS:BX - буфер
Структура буфера:
См. Длина Значение
00 2 0
02h 4 Серийный номер диска
06h 11 Метка тома или 'NONAME', если отсутствует
11h 8 Тип фата (AL=0) FAT12 или FAT16

Функция 6cH.

Функция расширенного открытия файла.
Вход:
AL - 00,
BX - режим открытия (см. ниже),
CX - атрибут файла (если файл создается),
DX - варианты действий:
OOxOH - выдать ошибку, если файла существует,
00x1H - открыть файл, если он существует,
OOx2H - обнулить файл (и открыть), если он существует,
000xH - выдать ошибку, если файл отсутствует,
001xH - создать файл, если он отсутствует.

Биты регистра BX:
0-2 - режим чтения-записи (0 - только для чтения, 1 - для записи,
2 - для чтения и записи).
4-6 - режим совместного доступа.

13 - 0 - нормальная обработка прерывания 24H, 1 - игнорировать прерывание 24H, но выдавать код ошибки не только при открытии и при дальнейших операциях чтения - записи.

14 - 0 обычная буферизация, 1 - не буферизованный ввод-вывод (безопасный).

Данная функция весьма напоминает функцию API CreateFile (Windows).

Приложение 8. Список функций BIOS.

Информация взята автором из различной справочной литературы, в том числе и компьютерного происхождения. Часть информации убрана мной из данной главы вследствие того, что она морально устарела. Часть информации оставлена мной не столько для практического использования, сколько в общеобразовательных целях. Описание функций BIOS для SVGA (по стандарту VESA) адаптеров помещено в главу 27. Сделано это для того, чтобы лишний раз подчеркнуть тот факт, что данная информация еще не устоялась.

Обслуживание видеосистемы (прерывание ЮН).

BIOS предоставляет пользователю ряд функций для управления дисплеем. Эти функции вызываются через прерывание 1ОН. Функции дисплея перечислены в таблице. Набор допустимых функций определяется типом дисплея и его адаптера, но функции с номерами 0 - 15 поддерживаются любым адаптером, допускающим графику. ПЗУ, содержащее видеофункции прерывания, ЮН находится на плате видеоадаптера и, естественно, корректно управляет видеосистемой. Это является одним из главных преимуществ использования 10-го прерывания. Недостатком является медленная их работа. Однако такие функции, как установка режима или переключение видеостраниц, будут полезны в любой программе.

номер	функция
0 (OOH)	установка режима дисплея
1 (01H)	установка типа курсора
2 (02H)	установка позиции курсора
3(O3H)	получить позицию и размер курсора
4 (04H)	получить позицию и состояние светового пера
5 (05H)	установка активной страницы дисплея
6 (06H)	скроллинг окна вверх
7 (07H)	скроллинг окна вниз
8 (08H)	чтение знака с атрибутом
9 (09H)	запись знака с атрибутом
10 (0AH)	запись знака
11 (0BH)	установка палитры
12(0CH)	чтение точки
13 (0DH)	записьточки
14 (0EH)	вывод знака в режиме телетайпа
15 (0FH)	получить режим дисплея

16 (10H)	установка регистров палитры
17 (11H)	установка знакогенератора
18 (12H)	дополнительные функции поддержки EGA
19 (13H)	вывод строки
20 (14H)	загрузка фонтов VGA
21 (15H)	получить физические параметры активного дисплея
26 (1AH)	получить или установить код типа дисплея
27 (1BH)	получение информации о состоянии
28 (1CH)	сохранение и восстановление состояния дисплея

Функции BIOS для управления дисплеем.

1. Установка режима дисплея

   Параметры: (AH) = 0,
   (AL) - режим дисплея
   Результатов нет.
   Вызов разрушает регистры AX, BP, SI, DI.

Допустимые режимы наиболее распространенных в ПП ЭВМ дисплеев (адаптеров) перечислены в таблице.

Режим (AL)	Тип	Разрешение	Поддерживается адаптерами	Цвет знак/фон	Адрес буфера
0	текстовый	40*25	все, кроме MDA	16 оттенков	B8000
1	текстовый	40*25	все, кроме MDA	16/8	B8000
2	текстовый	80*25	все, кроме MDA	16 оттенков	B8000
3	текстовый	80*25	все, кроме MDA	16/8	B8000
4	графический	320*200	все, кроме MDA	4	B8000
5	графический	320*200	все, кроме MDA	4 оттенка	B8000
6	графический	640*200	все, кроме MDA	2	B8000
7	текстовый	80*25	MDA	нет	B0000
8	графический	160*200	PCjr	16	B0000
9	графический	320*200	PCjr	16	B0000
10	графический	640*200	PCjr	16	B0000

11	резерв				
12	резерв				
13	графический	320*200	EGA,VGA	16	A0000
14	графический	640*200	EGA,VGA	16	A0000
15	графический	640*350	EGA,VGA	нет	A0000
16	графический	640*350	VGA	16	A0000
17	графический	640*480	VGA	2	A0000
18	графический	640*480	VGA	16	A0000
19	графический	320*200	VGA	256	A0000

Режимы адаптеров дисплея

Режимы дисплея подразделяются на текстовые и графические. В текстовых режимах буфер дисплея содержит коды символов, интерпретируемые знакогенератором, а также байты атрибутов, определяющие цвет и другие характеристики изображений знаков (мерцание, интенсивность, иногда подчеркивание). В текстовых режимах экран рассматривается как 25 строк текста по 40 или 80 знаков в строке.

В графических режимах возможен доступ к отдельным точкам (точнее, элементам изображения) экрана. Экран рассматривается как 200 (или 350) линий по 320 или 640 точек на каждой из них. Буфер дисплея содержит атрибут каждого элемента изображения. Вывод текстовой информации в графических режимах поддерживается специальными модулями BIOS.

В адаптере дисплея CGA имеется специальный порт управления режимом (3D8H). Запись байта в этот порт - режим экрана. В следующей таблице показана кодировка байта управления режимом.

Кодировка битов порта управления режимом дисплея CGA

режим	биты порта 3D8H								
	1	6	5	4	3	2	1	0	
0	X	x	b	0	c	1	0	0	x - неиспользуемый бит
1	x	x	b	0	c	0	0	0	b - бит мерцания:
2	x	x	b	0	c	1	0	1	0 - выключено,
3	x	x	b	0	c	0	0	1	1 - включено.
4	x	x	b	0	c	1	1	x	c - бит видимости:
5	x	x	b	0	c	x	x	x	0 - изображение не выводится на экран,
6	x	x	b	1	c	x	x	x	1 - изображение выводится на экран

Режимы 0, 2 и 5 аналогичны режимам 1, 3 и 4, за исключением того, что в первых подавляется цвет. Фактически подавление цвета возможно только в дисплеях определенного типа (так называемых составных мониторах); в наиболее распространенных мониторах (RGB), в которых каждый основной цвет управляется своим сигналом, такое подавление невозможно, так что режим 1, например, ничем не отличается от режима 0.

При установке режима экран очищается даже в случае, когда режим не изменяется. Вызов функции 0, однако, не лучший способ очистки экрана; функция 6 или 7 работает несколько быстрее. Некоторые адаптеры (EGA, PCjr, VGA) позволяют избежать очистки экрана при смене режима. Такой режим кодируется единицей в старшем бите (AL).

Отметим еще, что при изменении режима меняется и размер курсора, в частности, в графических режимах курсор не отображается на экране.

При включении питания или системном сбросе устанавливается режим дисплея, определяемый переключателями конфигурации (обычно режим 0), MS DOS при загрузке устанавливает режим 2. Текущая установка режима хранится в области данных BIOS и может быть прочитана, однако предпочтительно воспользоваться для получения режима функцией (AH)=15 INT 10H.

Замечания:

1. EGA, PCjr, PC Convertible, VGA. Если бит 7 (AL) = 1, то буфер экрана не очищается при установке режима.
2. EGA. Начальный режим (при включении питания) определяется переключателями в адаптере.
3. PC Convertible. Начальный режим при работе с цветным монитором - 2, с черно-белым - 7. Режим 7 используется как черно-белый для дисплея VGA (640*200) или для MDA (640*350).
4. VGA. Начальный режим дисплея - 3. Для всех режимов, кроме режима 19, инициализируются первые 16 регистров управления цветом; значения остальных 240 регистров управления цветом не определены.
5. VGA. Начальный режим дисплея 3 (цветовой дисплей) или 7 (черно-белый). Для всех режимов, кроме режима 19, инициализируются первые 64 регистра управления цветом; значения остальных 192 регистров управления цветом не определены.

2. Установка типа курсора

Параметры: (AH) = 1,
(CH) - номер верхней линии курсора,
(CL) - номер нижней линии курсора.

Результатов нет.

Вызов разрушает регистры AX, BP, SI, DI.

Вывод текстовой информации на экран средствами BIOS производится с позиции курсора. В текстовых режимах положение и размер курсора фиксируются во

внутренних регистрах адаптера дисплея (аппаратный курсор), в графических режимах - моделируются средствами BIOS. При выводе знака курсор не перемещается автоматически.

Для установки размера курсора используется функция 1 INT 10H. Размер и форма курсора определяются параметрами в четырех младших битах регистров CH и CL. Строки знакоместа на экране нумеруются сверху вниз, начиная с 0, и курсор представляет собой прямоугольник, состоящий из всех точек указанных линий. Если (CL) < (CH), то курсор будет состоять из двух частей.

Допустимые значения параметров CH и CL определяются типом адаптера и размером знакоместа на экране: для CGA знакоместо составляется из 8 строк, так что параметры не должны превышать 7, для MDA и EGA они не должны превышать 13. При включении питания или системном сбросе устанавливается размер курсора: (CH) = 6 и (CL) = 7 для CGA и EGA в режиме, отличном от режима MDA, (CH) = 11 и (CL) = 12 для MDA и EGA в режиме MDA.

Биты 5 и 6 (CH) в некоторых адаптерах можно использовать для указания частоты мерцания курсора, в частности, можно с их помощью сделать курсор невидимым. Однако использование этого средства нельзя рекомендовать, поскольку оно не будет работать на некоторых компьютерах. Проще сделать курсор невидимым, установив его (функцией 2 - см. ниже) за пределами экрана.

Функция 1 устанавливает размер курсора для всех страниц дисплея. Невозможно установить различные курсоры для разных страниц. В графических режимах нет аппаратного видимого курсора, и функция просто сообщает BIOS размер курсора, который будет использоваться, когда режим экрана станет текстовым.

Замечание. VGA. Перед записью в регистры адаптера значение (CH) удваивается, а (CL) удваивается и инкрементируется.

3. Установка позиции курсора.

Параметры: (AH) = 2,
(BH) - номер страницы дисплея,
(DH) - номер строки,
(DL) - номер столбца.
Результатов нет.
Вызов разрушает регистры AX, BP, SI, DI.

Функция устанавливает позицию курсора на указанной странице дисплея. Страница может быть активной, и тогда в текстовом режиме изображение курсора переместится на экране, или неактивной. Для графических режимов номер страницы должен быть равен 0. Допустимые номера страниц для текстовых режимов приведены в таблице ниже. Корректность номера страницы не проверяется, и использование неверного номера приводит к непредсказуемым результатам.

Координаты курсора во всех режимах задаются в текстовых единицах. Строки экрана нумеруются сверху вниз от 0 до 24, а столбцы - слева направо от 0 до 40 или

80. Допускается (DH)=25, такое указание выводит курсор за пределы видимого экрана.

В графических режимах вызов функции 2 сообщает BIOS позицию, с которой будет выведен следующий текстовый знак. Поскольку в этих режимах нет аппаратного курсора, не будет никакого изменения изображения на экране. Программы, использующие (видимый) курсор в графических режимах, моделируют его изображение.

Режим	Размер экрана	Текстовый размер	Размер знака	Число страниц	Адаптер
0, 1	320*200	40*25	8*8	8	кроме MDA
	320*350		8*14	8	EGA, VGA
	320*400		8*16	8	
	360*400		9*16	8	VGA
2, 3	640*200	80*25	8*8	4	PC-jr,CGA, PC Convertible
	640*200		8*8	8	EGA, VGA
	640*350		8*14	8	EGA, VGA
	640*400		8*16	8	VGA
	720*400		9*16	8	VGA
4, 5	320*200	40*25	8*8	1	кроме MDA
6	640*200	80*25	8*8	1	кроме MDA
7	720*350	80*25	9*14	1	MDA, PC Convert
	720*350		9*14	8	EGA, VGA
	720*400		9*16	8	VGA
 8	640*200 160*200	 20*25	8*8 8*8	4 1	PC Convertible PC-jr
9	160*200	40*25	8*8	1	PC-jr
10	160*200	80*25	8*8	1	PC-jr
13	320*200	40*25	8*8	8	EGA, VGA
14	640*200	80*25	8*8	4	EGA, VGA
5,16	640*350	80*25	8*14	2	EGA, VGA
17	640*480	80*30	8*16	1	VGA
18	640*480	80*30	8*16	1	VGA
19	320*200	40*25	8*8	1	VGA

Аппаратные спецификации режимов дисплея

4. Получить позицию и размер курсора

Параметры: (AH) = 3,
(BH) - номер страницы дисплея.
Результат: (CH) - номер верхней линии курсора,
(CL) - номер нижней линии курсора,
(DH) - номер строки,
(DL) - номер столбца.
Вызов разрушает регистры AX, BP, SI, DI.

Функция возвращает размер и положение курсора в той же форме, которая требуется для установки этих значений функциями 1 и 2. Страница, для которой возвращается позиция курсора, может и не быть активной. Для графических режимов номер страницы должен быть равен 0. Допустимые номера страниц для текстовых режимов приведены в 11.5. Корректность номера страницы не проверяется, и использование неверного номера приводит к непредсказуемым результатам.

5. Получить позицию и состояние светового пера

Параметры: (AH) = 4.
Результат: (AH) - состояние светового пера:
0 - неактивно,
1 - активно, и
(BX) - координата пера по горизонтали;
(CH) - координата пера по вертикали (для режимов 15 - 18 - (CX));
(DH) - номер строки;
(DL) - номер столбца.

Вызов разрушает регистры AX, BP, SI, DI. При (AH) = 0 содержимое регистров BX, CX и DX не определено.

Координаты светового пера возвращаются в двух формах: позиция знакоместа (номер строки и номер столбца) и координаты на графическом экране. Графические координаты неточны: вертикальная координата всегда четна, а горизонтальная делится на 4 или на 8 в зависимости от режима экрана.

Некоторые компьютеры возвращают (AH) = 0 как признак того, что световое перо не установлено.

Замечание. PC Convertible, VGA-функция не поддерживается; всегда возвращается (AH) = 0; (BX), (CX), (DX) разрушается.

6. Установка активной страницы дисплея

Параметры: (AH) = 5,
(AL) - номер страницы.
Результатов нет.
Вызов разрушает регистры AX, BP, SI, DI.

Номер страницы отсчитывается от нуля. Число страниц зависит от режима дисплея и типа адаптера - см. табл. 5. Наличие страниц функцией не проверяется.

Замечание. PC-jr. Поддерживаются четыре подфункции:
(AL) = 80H - читать регистры страницы. Выход:
(BL) - регистр страницы микропроцессора,
(BH) - регистр страницы дисплея;
(AL) = 81H - установить регистр страницы микропроцессора.
Вход:
(BL) - номер страницы;
(AL) = 82H - установить регистр страницы дисплея. Вход:
(BL) - номер страницы;
(AL) = 83H - установить регистры страницы. Вход:
(BL) - регистр страницы микропроцессора,
(BH) - регистр страницы дисплея.

7. Скроллинг окна вверх

Параметры: (AH) = 6,
(AL) - величина сдвига,
(BH) - атрибут заполнителя пустой строки,
(CH) - номер строки левого верхнего угла окна,
(CL) - номер столбца левого верхнего угла окна,
(DH) - номер строки правого нижнего угла окна,
(DL) - номер столбца правого нижнего угла окна.
Результатов нет.
Вызов разрушает регистры AX, BP, SI, DI.

Все параметры задаются в текстовой форме даже для графических режимов. Прямоугольное окно экрана, определенное координатами (CX) и (DX), перемещается вверх на (AL) строк. Информация, выходящая за пределы экрана, теряется, а освобождаемая область заполняется знаком пробела (код 20H) с атрибутом (BH). В графических режимах область заполняется нулевыми кодами. При (AL) = 0 окно очищается.

Замечание: фактически координатами левого верхнего угла являются min {(CH),(DH)} и min{(CL),(DL)}, а координатами верхнего правого угла - max {(CH),(DH)} и max {(CL),(DL)}.

8. **Скроллинг окна** вниз

Параметры: (AH) = 7,
(AL) - величина сдвига,
(BH) - атрибут заполнителя пустой строки,
(CH) - номер строки левого верхнего угла окна,
(CL) - номер столбца левого верхнего угла окна,
(DH) - номер строки правого нижнего угла окна,
(DL) - номер столбца правого нижнего угла окна.
Результатов нет.
Вызов разрушает регистры AX, BP, SI, DI.

Функция совершенно аналогично функции (AL) = 6 скроллинга вверх.

9. Чтение знака и атрибута

Параметры: (AH) = 8,
(BH) - номер страницы дисплея.
Результат: (AL) - код знака,
(AH) - атрибут знака.
Вызов разрушает регистры BP, SI, DI.

Функция возвращает код и атрибут знака в позиции курсора на указанной странице дисплея. Страница не обязана быть активной. Для графических режимов необходимо (BH) = 0.

В текстовых режимах дисплея в его буфере хранятся коды и атрибуты знаков, а изображения знаков вырабатываются знакогенератором дисплея. Знакоместу экрана соответствуют два смежных байта в памяти дисплея, первый из которых содержит код знака, второй - его атрибут. Если n - номер строки, m - номер столбца и p - номер страницы, то адрес кода знака в буфере дисплея CGA определяется формулами:

для режимов 0 и 1 B8000H+2048*p+80*n+2*m;

для режимов 2 и 3 B8000H+4096*p+160*n+2*m.

Атрибуты знака кодируются в следующем байте памяти: биты 2 - 0 - цвет знака:
000 - черный,
001 - синий,
010 - зеленый,
011 - голубой,
100 - красный,
101 - оранжевый,
110 - коричневый,
111 - белый;
бит 3 - бит интенсивности: 0 - нормальная интенсивность, 1 - высокая интенсивность; биты *6-4*- цвет фона (прямоугольного окна, в котором изображается знак);

кодируется аналогично цвету знака; бит 7 - бит мерцания: 0 - нормальное изображение, 1 - мерцающее изображение.

В цветовых текстовых режимах атрибут прямо используется для управления изображением на экране; в режимах подавления цвета любой код, отличный от 000, дает белый цвет (см., однако, замечание о типах дисплеев в п. 11.1).

Другие адаптеры используют сходную кодировку байта атрибутов, но могут допускать другое использование битов 3 и 7 и программируемый выбор набора цветов знака и фона.

В графических режимах используется совершенно другая кодировка информации (см. 11.12); когда функция (AL) = 8 применяется в графических режимах, она использует для распознавания знаков (знакоместо может и не содержать кода никакого знака) таблицу изображений знаков. Изображения знаков с кодами 0 - 7FH хранятся в ПЗУ BIOS; таблица изображений остальных знаков (80H - OFFH) указывается вектором 1FH. BIOS не устанавливает этот вектор, точнее, устанавливает его на некоторый случайный адрес ПЗУ. Для использования в графических режимах знаков с кодами 80H - OFFH необходима загрузка специальной программы GRAFTABL[69], которая резидентно загружает таблицу изображений дополнительных знаков и устанавливает вектор 1FH на ее начало.

Если функция 8 не распознает графического изображения знака, то возвращается (AL) = 0.

10. Запись знака с атрибутом

Параметры: (AH) = 9,
(AL) = код знака,
(BH) - номер страницы дисплея,
(BL) - атрибут знака,
(CX) - счетчик повторений.
Результатов нет.
Вызов разрушает регистры AX, BP, SI, DI.

Функция копирует знак и его атрибут в память дисплея. Если (BH) указывает на активную страницу, то изображение знака немедленно появится на экране.

Регистр CX указывает число повторений вывода знака. Следует отметить, что для вывода одного знака нужно указать (CX) = 1; при (CX) = 0 выводится до 65536 знаков. Знаки выводятся в последовательные адреса, начиная с адреса, соответствующего позиции курсора на странице. Позиция курсора при выводе не изменяется. Если при выводе в графическом режиме установить бит 7 BL в единицу, то атрибут знака будет поразрядно складываться с текущим атрибутом знакоместа экрана (инструкцией XOR).

Для режима 19 в (BH) передается цвет фона.

Вывод знака с одновременным изменением позиции курсора выполняется функцией 14.

[69] Хорошие русификаторы обрабатывают и эти режимы.

11. Запись знака без изменения атрибута

Параметры: (AH) =10,
(AL) = код знака,
(BH) - номер страницы дисплея,
(CX) - счетчик повторений.
Результатов нет.
Вызов разрушает регистры AX, BP, SI, DI.

Функция аналогична предыдущей, но атрибуты знаков в памяти дисплея не заменяются. Не рекомендуется пользоваться этой функцией в графических режимах.

12. Установка палитры

Параметры: (AH) =11,
(BH) - код подфункции (0 или 1),
(BL) - номер основного цвета или номер палитры.
Результатов нет.
Вызов разрушает регистры AX, BP, SI, DI.

Подфункция (BH) = 0 применима для любого режима дисплея. В текстовых режимах она устанавливает цвет бордюра экрана; (BL) может принимать любое значение от 0 до 16. В графических режимах подфункция не только определяет цвет бордюра, но также в режимах среднего разрешения (320*200) задает основной цвет. Этому цвету соответствует в выбранной палитре (см. ниже) код 0. В графических режимах высокого разрешения (где допускается только два цвета) подфункция 0 устанавливает цвет фона и цвет бордюра.

Подфункция (BH) = 1 используется для установки палитры (цветового набора) в графических режимах. Для адаптера CGA допускается ее использование только в режимах 4 и 5, для других адаптеров (EGA, PCjr) она может применяться и в других режимах. В (BL) задается код палитры (0 или 1 для адаптера CGA).

Каждая палитра состоит из четырех цветов.

Для палитры 0:
0 - основной цвет (устанавливаемый отдельно),
1 - зеленый,
2 - красный,
3 - коричневый.

Для палитры 1:
0 - основной цвет,
1 - синий,
2 - голубой,
3 - белый.

Замечание для EGA, VGA. Если функция выполняется в графическом режиме 640*400 и (BH) = 0, то устанавливается цвет фона.

13. **Запись точки**

Параметры: (AH)=12,
(AL) - атрибут (цвет) точки,
(CX) - позиция по горизонтали,
(DH) - позиция по вертикали.
Результатов нет.
Вызов разрушает регистры AX, BP, SI, DI.

Функция применима только в графических режимах.

Атрибутом точки является код ее цвета: в режимах среднего разрешения код цвета согласно выбранной палитре, в режимах высокого разрешения 0 обозначает белый (или основной) цвет, 1 - черный.

Буфер графического дисплея заполнен атрибутами точек, причем первые 8 Кбайт соответствуют линиям сканирования экрана с четными номерами (0, 2, ..., 198), следующие 8 Кбайт - линиям с нечетными номерами. В режимах среднего разрешения точке экрана соответствуют два бита буфера, в режимах высокого разрешения - один бит. Позиция атрибута точки с горизонтальной координатой m и вертикальной координатой m определяется следующим образом:

- для режимов среднего разрешения -
  биты 2*(m mod 4) и 2*(т mod 4)+1 байта с адресом

```
B8000H+8192*(n mod 2)+80*(n/2)+m/4
```

- для режимов высокого разрешения - бит 7-(m mod 8) байта с адресом

```
B8000H+8192*(n mod 2)+80*(n/2)+m/8
```

Выполнение функции зависит от старшего бита (AL): если он равен 0, то атрибут точки просто помещается в буфер дисплея, если же этот бит равен 1, то выполняется поразрядное сложение (операцией XOR) атрибута с соответствующими битами буфера дисплея.

Замечание: для адаптеров, поддерживающих несколько графических страниц, номер страницы указывается в (BH).

14. Чтение точки

Параметры: (AH)=13,
(CX) - позиция по горизонтали,
(DH) - позиция по вертикали.
Результат: (AL) - атрибут (цвет) точки,
Вызов разрушает регистры AX, BP, SI, DI.

Функция применима только в графических режимах и возвращает атрибут точки графического дисплея в младших битах (AL). Старшие биты (AL) сбрасываются в 0.

Замечание: для адаптеров, поддерживающих несколько графических страниц, номер страницы указывается в (BH).

15. Вывод знаков в режиме телетайпа

Параметры: (AH) = 14,
(AL) - знак для вывода,
(BL) - атрибут знака (используется только в графических режимах).
Результатов нет.

Знак выводится в текущую позицию курсора на активной странице, и курсор продвигается с учетом возможного перехода на новую строку и скроллинга при достижении конца страницы.

Знаки "возврат каретки" (ODH), "перевод строки" (0AH), "звонок" (07) и "возврат на шаг" (08) обрабатываются особо и не появляются на экране.

Замечания: в некоторых моделях компьютера требуется указание номера активной страницы в BH.

16. Получить режим дисплея

Параметры: (AH) =15.
Результат: (AH) - ширина экрана в текстовом формате;
(AL) - режим дисплея;
(BH) - номер активной страницы.
Вызов разрушает регистры BP, SI, DI.

В графических режимах возвращается (BH) = 0.

17. Дополнительные средства управления цветом

Четыре следующие подфункции применимы только для адаптеров EGA, PCjr, VGA.

1. Установка регистра палитры.

Параметры: (AX) = 1000H,
(BL) - номер регистра,
(BH) - значение для установки.
Результатов нет.

Каждый из 16 регистров управления палитрой может принимать одно из 64 значений. Биты (BH) кодируются следующим образом:

бит 0 - голубая компонента нормальной интенсивности,
бит 1 - зеленая компонента нормальной интенсивности,

бит2 - красная компонента нормальной интенсивности,
бит 3 - голубая компонента меньшей интенсивности,
бит4 - зеленая компонента меньшей интенсивности,
бит 5 - красная компонента меньшей интенсивности,
биты 6 и 7 резервируются.

Нормальный основной цвет получается установкой нормальной компоненты, интенсивный основной цвет - установкой обеих компонент, но допускается любая комбинация бит. Значения регистров управления палитрой определяют набор используемых оттенков. Атрибут знака и фона используется как индекс регистра. Так, адаптер EGA определяет следующие умолчания для значений регистров управления палитрой:

регистр	код	цвет
0	OOH	черный
1	01H	синий
2	02H	зеленый
3	O3H	голубой
4	04H	красный
5	05H	оранжевый
6	14H	коричневый
7	07H	белый
8	38H	темно-серый
9	39H	светло-голубой
10	3AH	светло-зеленый
И	3BH	светло-синий
12	3CH	розовый
13	3DH	палевый
14	OEH	желтый
15	3FH	интенсивный белый

Умолчания для регистров управления палитрой адаптера EGA

2. Установка цвета бордюра.

Параметры: (AX) = 1001H,
(BH) - значение для установки.

Результатов нет.

3. Установка регистров палитры и цвета бордюра.

Параметры: (AX) = 1002H,
(ES:DX) - указатель на 17-байтовый блок данных,
первые 16 байтов которого содержат значения, посылаемые
в регистры палитры, а последний байт - цвет бордюра.
Результатов нет.

4. Выбор интенсивности или мерцания.

Параметры: (AX) = 1003H,
(BL) = 0 - разрешение интенсивности,
(BL) = 1 - разрешение мерцания.

Функция поддерживается для адаптеров, в которых интесивность цвета и мерцание знаков управляются одним и тем же битом в регистре адаптера.

Следующие функции применимы только в VGA.

5. Чтение регистра палитры.

Параметры: (AX) = 1007H,
(BL) - номер регистра палитры (0 - 15).
Результат: (BH) - прочитанное значение.

6. Чтение регистра бордюра.

Параметры: (AX) = 1008H.
Результат: (BH) - прочитанное значение.

7. Чтение регистров палитры и бордюра.

Параметры: (AX) = 1009H,
(ES:DX) - указатель 17-байтовой области для результатов.
Результат: байты 0 - 15 области результатов - значения регистров палитры;
байт 16 - значение регистра бордюра.

8. Установка регистра управления цветом.

Параметры: (AX) = 1010H,
(BX) - номер регистра,
(DH) - значение красной компоненты,
(CH) - значение зеленой компоненты,
(CL) - значение голубой компоненты.
Результатов нет.

9. Установка блока регистров управления цветом.

Параметры: (AX) = 1012H,
(ES:DX) - указатель таблицы значений цветовых компонент;
(BX) - номер первого из регистров управления цветом,
(CX) - число устанавливаемых регистров.
Результатов нет.

Таблица значений содержит три байта для каждого из устанавливаемых регистров: значения красной, зеленой и голубой компонент цвета.

10. Выбор режима управления цветом.

Параметры: (AX) = 1013H,
(BL) = 0,
(BH) - режим:
0 - устанавливается 4 блока по 64 регистра управления цветом;
1 - устанавливается 16 блоков по 16 регистров управления цветом.
Результатов нет.
Функция недопустима в режиме дисплея 19.

11. Выбор палитры.

Параметры: (AX) = 1013H,
(BL) = 1,
(BH) - номер блока регистров управления цветом
(0 - 3 или 0 - 15 в зависимости от ранее выбранного режима).
Результатов нет.

Замечание. Функция (AH) = 0 устанавливает умолчания для первых 64 регистров управления цветом и 64-регистровые блоки. Значения регистров в трех других блоках не определены.

12. Чтение регистра управления цветом.

Параметры: (AX) = 1015H,
(BX) - номер регистра,
Результаты: (DH) - значение красной компоненты,
(CH) - значение зеленой компоненты,
(CL) - значение голубой компоненты.

13. Чтение блока регистров управления цветом.

Параметры: (AX) = 1017H,
(ES:DX) - указатель таблицы значений цветовых компонент;

(BX) - номер первого из регистров управления цветом,
(CX) - число регистров
Результаты: таблица значений содержит три байта для каждого из устанавливаемых регистров: значения красной, зеленой и голубой компонент цвета.

14. Чтение режима управления цветом.

Параметры: (AX) = 101AH,
Результаты: (BL) - режим управления цветом (0 или 1),
(BH) - номер активного блока регистров управления цветом.

15. Суммирование компонент цвета в оттенки серого.

Параметры: (AX) = 101BH,
(BX) - номер первого из регистров управления цветом
(CX) - число регистров.
Результатов нет.

Функция читает компоненты цвета из указанных регистров, выполняет взвешенное суммирование (30% красной компоненты, 59% зеленой и 11% голубой) и записывает результат в эти регистры.

Следующие функции применимы в VGA:
(AX) = 1000H, (BX) = 0712H - установка регистров управления цветом соответственно восьми основным цветам; результатов нет;
(AX) = 1003H - см. 11.17.4;
(AX) = 1010H - см. 11.17.8;
(AX) = 1012H - см. 11.17.9;
(AX) = 1015H-см. 11.17.12;
(AX) = 1017H - см. 11.17.13;
(AX) = 101BH - см. 11.17.15.

18. Установка знако-генератора

Следующие подфункции функции (AH) = 11H позволяют пользователю определять изображения наборов знаков в текстовых и графических режимах дисплея. Эти функции поддерживаются только адаптером EGA, и допустимые значения параметров определяются объемом буфера адаптера.

Каждые 64 Кбайта буферной памяти позволяют определить один блок изображений знаков. Таким образом, допускается от одной до четырех таблиц изображений, каждая из которых хранит изображения 256 знаков. По умолчанию все блоки инициируются одной и той же стандартной таблицей. В режимах 4 - 6 адаптер использует изображения только первых 128 знаков, в других режимах - изображения всех 256 знаков.

1. Загрузка таблицы знакогенератора (текстовые режимы).

   Параметры: (AX) = 1100H,
   (ES:BP) - указатель на таблицу изображений знаков,
   (CX) - число знаков,
   (DX) - смещение в таблице знаков,
   (BL) - номер блока (таблицы знакогенератора),
   (BH) - длина изображения знака (байт).
   Результатов нет.

2. Установка стандартной монохромной таблицы.

   Параметры: (AX) = 1101H,
   (BL) - номер блока (таблицы знакогенератора).
   Результатов нет. Загружается таблица фонтов 8* 14.

3. Установка стандартной цветовой таблицы.

   Параметры: (AX) = 1102H,
   (BL) - номер блока (таблицы знакогенератора).
   Результатов нет. Загружается таблица фонтов 8*8.

4. Установка указателя блока для текстовых режимов.

   Параметры: (AX) = 1103H,
   (BL) - выбор блоков:
   биты 3 - 2 - номер блока для использования в случае, когда бит 3 байта атрибутов знака равен 1,
   биты 1 - 0 - номер блока для использования в случае, когда бит 3 байта атрибутов знака равен 0.
   Результатов нет.

Вызовы функций 11OOH - 1102H завершают установку (текстового) режима EGA, позволяя, кроме того, определить таблицу пользователя изображений знаков. В сочетании с функцией 1103H они позволяют использовать набор из 512 знаков.

Если, например, пользователь хочет работать с набором знаков блока 3, он может вызвать функцию 1103H с (BL) = OFH. Будет установлен набор 3, а бит 3 байта атрибутов будет управлять интенсивностью знака (см. 16.11). Вообще если при вызове функции 1103H бит 2 (BL) равен биту 0 и бит 3 равен биту 1, то бит 3 байта атрибутов будет управлять интенсивностью.

Вызов функции 1103H, например, с (BL) = OCH, позволяет определить набор из 512 знаков. Если бит 3 байта атрибутов знака равен 0, то изображение знака будет выбираться из блока 0, если бит 3 равен 1 - то из блока 1. Если используется набор из

512 знаков, то рекомендуется вызвать функцию 1000Н с (ВХ) = 0712Н для установки подходящей цветовой палитры.

5. Подфункции (АХ) = 111OН, 1111Н и 1112Н аналогичны соответственно подфункциям (АХ) = 1100Н, 1101Н и 1102Н со следующими отличиями:
   - они могут использоваться только непосредственно после установки режима дисплея (функцией 0);
   - при вызове должна быть активна страница 0;
   - при вызове переопределяется число байтов на знак, число строк экрана и длина буфера и соответственно программируются регистры адаптера.

   Эти подфункции также применимы только в текстовых режимах.

6. Загрузка дополнительной таблицы знаков графики.

   Параметры: (АХ) = 1120Н,
   (ES:BP) - указатель таблицы изображений знаков с кодами 80Н - OFFH (в формате 8*8 точек).
   Результатов нет.

7. Загрузка таблицы знакогенератора (графические режимы).

   Параметры: (АХ) = 1121Н,
   (ES:BP) - указатель таблицы изображений знаков,
   (СХ) - длина изображения знака (байт),
   (BL) - число строк текстового экрана:
   О - число строк указано в (DL),
   1 - 14 строк,
   2 - 25 строк,
   3 - 43 строки.
   Результатов нет.

8. Установка стандартной таблицы знаков размера 8* 14 точек.

   Параметры: (АХ) = 1122Н,
   (BL) - число строк текстового экрана (как в функции 1121Н).
   Результатов нет.

9. Установка стандартной таблицы знаков размера 8*8 точек.

   Параметры: (АХ) = 1123Н,
   (BL) - число строк текстового экрана (как в функции 1121Н).
   Результатов нет.

Таблица пользователя для подфункции 1121Н должна определять все 256 знаков с кодами 00 - OFFH, а указатель таблицы для подфункции 1120Н должен совпадать с

вектором прерывания 1FH. Как и подфункции 11ЮН - 1112H, данные подфункции программируют регистры адаптера и могут быть вызваны только непосредственно после установки режима дисплея.

10. Получить информацию о знакогенераторе.

Параметры: (AX) = 1 BOH,
(BH) - код возвращаемого указателя:
0 - указатель INT 1FH,
1 - указатель INT 44H,
2 - указатель стандартной таблицы формата 8*14 (ПЗУ),
3 - указатель стандартной таблицы формата 8*8 (ПЗУ),
4 - указатель вершины стандартной таблицы формата 8*8,
5 - указатель альтернативной таблицы формата 9*14 (ПЗУ).
Результат: (CX) - длина изображения знака (байт),
(DL) - число строк текстового экрана,
(ES:BP) - указатель на таблицу изображений знаков.

Следующие подфункции применимы в VGA:
подфункции (AX) = 1100H, 1101H, 1102H (см. 11.18.1 - 11.18.3);
подфункции (AX) = 1120H, 1121H, 1122H, 1123H (см. 11.18.6 - 11.18.9);

11. Установка указателя блока для текстовых режимов.

Параметры: (AX) = 1103H,
(BL) - выбор блоков:
биты 4,1,0 - номер блока для использования в случае, когда бит 3 байта атрибутов знака равен О,
биты 5,3,2 - номер блока для использования в случае, когда бит 3 байта атрибутов знака равен 1.
Результатов нет.

Вызовы функций 1100H - 1102H завершают установку (текстового) режима, позволяя, кроме того, определить таблицу пользователя изображений знаков. В сочетании с функцией 1103H они позволяют использовать набор из 512 знаков.

Если, например, пользователь хочет работать с набором знаков блока 6, он может вызвать функцию 1103H с (BL) = 03AH. Будет установлен набор 6, а бит 3 байта атрибутов будет управлять интенсивностью знака (см. 16.11). Вообще если при вызове функции 1103H бит 2 (BL) равен биту 0, бит 3 равен биту 1 и бит 5 равен биту 4, то бит 3 байта атрибутов будет управлять интенсивностью.

Вызов функции 1103H, например, с (BL) = 28H, позволяет определить набор из 512 знаков. Если бит 3 байта атрибутов знака равен 0, то изображение знака будет выбираться из блока 0, если бит 3 равен 1, то - из блока 6. Если используется набор из

512 знаков, то рекомендуется вызвать функцию 1000H с (BX) - 0712H для установки подходящей цветовой палитры.

12. Установка стандартной таблицы фонтов 8*16.

    Параметры: (AX) = 1104H,
    (BL) - число строк текстового экрана (как в функции 1121H).
    (BL) - номер блока (таблицы знакогенератора).
    Результатов нет. Загружается таблица фонтов 8*16.

13. Подфункции (AX) = 11ЮН - 1112H и 1114H аналогичны соответственно подфункциям (AX) = 1100H - 1102H и 1104H со следующими отличиями:
    - они могут использоваться только непосредственно после установки режима дисплея (функцией 0);
    - при вызове должна быть активна страница 0;
    - при вызове переопределяется число байтов на знак, число строк экрана и длина буфера и соответственно программируются регистры адаптера.

    Эти подфункции также применимы только в текстовых режимах.

14. Установка стандартной таблицы фонтов 8*16.

    Параметры: (AX) = 1124H,
    (BL) - число строк текстового экрана (как в функции 1121H).
    Результатов нет. Загружается таблица фонтов 8*16.

15. Получить информацию о знакогенераторе.

    Параметры: (AX) = 1130H,
    (BH) - код возвращаемого указателя:
    0 - указатель INT 1FH,
    1 - указатель INT 44H,
    2 - указатель стандартной таблицы формата 8*14 (ПЗУ),
    3 - указатель стандартной таблицы формата 8*8 (ПЗУ),
    4 - указатель вершины стандартной таблицы формата 8*8,
    5 - указатель альтернативной таблицы формата 9*14 (ПЗУ),
    6 - указатель таблицы 8*16 ПЗУ,
    7 - указатель альтернативной таблицы 9*16.

    Результат: (CX) - длина изображения знака (байт),
    (DL) - число строк текстового экрана,
    (ES:BP) - указатель на таблицу изображений знаков.

Следующие подфункции применимы в VGA:
(AX) = 1100H, 1102H, 1103H (см. 11.18.1, 11.18.3, 11.8.4);
(AX) = 1104H (см. 11.18.12);

(AX) = 1120H, 1121H, 1123H(см. 11.18.6, 11.18.7, 11.18.9);
(AX) =1124, 1130H (см. 11.18.14 и 11.18.15).

Замечания. VGA. Функции 1101H, 1110H -1114H, 1122H, если вызываются, эквивалентны соответственно 1104H, 1100H - 1104H, 1124H. Если в функции 1100H указано (BH) = 14, то 14-байтовые знаки расширяются двумя пустыми последними строками до 16-байтовых. Для функции 113OH недопустимы значения (BH) =. 5 или 7.

19. Дополнительные функции поддержки адаптера EGA

Две следующие подфункции применимы только для адаптеров EGA и в VGA.

1. Получить информацию об установке адаптера.

Параметры: (AH) = 12H,
(BL) = 10H.
Результат: (BH) = 0 - установлен режим цветового адаптера (порты 3DxH),
1 - монохромный режим (порты 3BxH),
(BL) - объем буфера:
O - 64 Кбайта,
1 - 128 Кбайт,
2 - 192 Кбайта,
3 - 256 Кбайт,
(CH) - биты адаптера (имеют специальное применение),
(CL) - установка переключателей на плате адаптера.

2. Установить альтернативный обработчик печати экрана.

Параметры: (AH) = 12H,
(BL) = 20H.
Результатов нет.

Данная функция предназначена для установки обработчика прерывания 5 (печать экрана), корректно работающего при изменении числа текстовых строк дисплея.

Остальные подфункции данного раздела применимы только в VGA.

3. Установить число скан-линий для текстового режима.

Параметры: (AH) = 12H,
(BL) = 30H,
(AL) = 0 - 200 скан-линий,
1 - 350,
2-400.

Результат: (AL) = 12H как признак того, что функция поддерживается. Эффект функции проявится только после установки текстового режима. Функция не поддерживается в модели 30.

4. Разрешение или запрет загрузки регистров палитры при смене режима

Параметры: (AH) = 12H,
(BL) = 31H,
(AL) = 0 - разрешение загрузки регистров палитры,
1 - запрет.
Результат: (AL) = 12H, как признак того, что функция поддерживается.

Если загрузка запрещена, то при последующей замене режима EGA останутся 16 регистров палитры, регистр бордюра и 256 регистров цвета.

5. Разрешение или запрет дисплея.

Параметры: (AH) = 12H,
(BL) = 32H,
(AL) = 0 - разрешение изображения,
1 - запрет.
Результат: (AL) = 12H, как признак того, что функция поддерживается.

Разрешается или запрещается доступ активного дисплея к буферу регенерации и к портам ввода-вывода.

6. Разрешение или запрет суммирования интенсивности цвета при смене режима

Параметры: (AH) = 12H,
(BL) = 33H,
(AL) = 0 - разрешение суммирования,
1 - запрет.
Результат: (AL) = 12H как признак того, что функция поддерживается.

Если суммирование разрешено, то оно будет выполняться при вызове функций (AH) = 0 (установить режим дисплея) и (AH) = ЮН (установить регистры палитры).

7. Разрешение или запрет курсора.

Параметры: (AH) = 12H,
(BL) = 34H,
(AL) = 0 - разрешение курсора,
1 - запрет.
Результат: (AL) = 12H как признак того, что функция поддерживается.

Если курсор разрешен, то размер курсора, переданный при выполнении функции (AH) = 1, будет масштабироваться при смене режима. Разрешение курсора устанавливается при включении питания. Функция не поддерживается в модели 30.

8. Функции переключения дисплея.

Параметры: (AH) = 12H,
(AL) - код подфункции:
0 - запрет доступа к дисплею со стороны адаптера,
1 - разрешение доступа к дисплею со стороны системы,

2 - переключение в активное состояние,
3 - переключение в пассивное состояние.
(ES:DX) - указатель на 128-байтовую область сохранения
(не используется при (AL) = 1).
Результат: (AL) = 12H как признак того, что функция поддерживается.

Четыре перечисленные подфункции позволяют избежать конфликтов аппаратной несовместимости и перекрытия области данных BIOS при несоответствии между системным управлением дисплеем и адаптером дисплея. Если доступ со стороны системы и со стороны адаптера к портам ввода-вывода, к буферу дисплея и, возможно, к области данных BIOS не приводит к каким-либо конфликтам, то и нет необходимости в применении этих функций. Перед их применением должна быть вызвана функция (AH) = 12H, (BL) = 32H, чтобы запретить изображение на экране.

При конфликтах доступа к оборудованию или к памяти со стороны адаптера и со стороны системной платы приоритет имеет адаптер. Функции системы остаются запрещенными до тех пор, пока дисплей разрешен для адаптера.

Переключение дисплея выполняется следующим образом:

1. Инициируется запрещение дисплея для адаптера (AL) = 0;
2. Инициируется доступ к дисплею со стороны системной платы.

Эти функции можно корректно вызвать лишь один раз. После их вызова становятся доступными функции, переключающие дисплей в активное ((AL) = 2) и в пассивное ((AL) = 3) состояние. При переключении в активное состояние запрещаются выполнение текущей функции и вывод изображения. Состояние дисплея сохраняется в указанной области. При переключении в пассивное состояние информация из буфера сохранения используется для восстановления состояния дисплея.

9. Разрешение или запрет изображения.

Параметры: (AH) = 12H,
(BL) = 36H,
(AL) = 1 - запретить вывод изображения,
0 - разрешить изображение.
Результат: (AL) = 12H как признак того, что функция поддерживается.
Функция не поддерживается в модели 30 System/2.

11.20. Вывод строки

Параметры: (AH) = 13H,
(AL) - код подфункции:
0 - вывод с атрибутом (BL) без перемещения курсора,
1 - вывод с атрибутом (BL) с перемещением курсора,
2 - вывод с указанными атрибутами без перемещения курсора,
3 - вывод с указанными атрибутами и перемещением курсора;
(BH) - номер страницы дисплея,
(BL) - атрибут для вывода знаков (при (AL) = 0 или 1),
(CX) - длина строки знаков (байт),
(DH) - номер строки для первого знака,

(DL) - номер столбца для первого знака,
(ES:BP) - указатель строки знаков.

Результатов нет.

Для подфункций 2 и 3 атрибут каждого знака следует за знаком в строке. Подфункции 0 и 2 не изменяют позиции курсора, подфункции 1 и 3 устанавливают курсор за последний выведенный знак. Подобно функции 14, данная функция особым образом обрабатывает знаки "возврат каретки" (ODH), "перевод строки" (OAH), "звонок" (07) и "возврат на шаг" (08); остальные знаки изображаются на экране.

Функция 13H поддерживается не во всех компьютерах, и ее поддержка зависит не от типа дисплея, а от версии BIOS.

21. Загрузка знакогенератора VGA

1. Загрузка фонтов пользователя.

Параметры: (AX) = 1400H,
(ES:DI) - указатель на первый знак в таблице пользователя изображений знаков,
(CX) - число изображений знаков в таблице (1 - 256),
(DX) - смещение знака в области для знакогенератора (ОЗУ),
(BL) = 0 - загружается основная таблица,
1 - загружается альтернативная таблица, иначе - пустая функция;
(BH) - длина изображения знака в байтах.

Результатов нет.

2. Загрузка системных фонтов из ПЗУ

Параметры: (AX) = 1401H,
(BL) = 0 - загружается основная таблица,
1 - загружается альтернативная таблица, иначе - пустая функция;

Результатов нет.

3. Управление битом интенсивности VGA.

Параметры: (AX) = 1402H,
(BL) = 0 - игнорировать бит интенсивности,
1 - использовать как бит обратной контрастности,
2 - использовать как бит подчеркивания,
3 - использовать как бит выбора альтернативной таблицы изображений, иначе - пустая функция.

22. Получение физических параметров активного дисплея

Параметры: (AH) = 15H,
Результаты: (AX) - тип альтернативного адаптера:
O - нет альтернативного адаптера,
5140 - VGA,

5153 - типа CGA,
5151 - типа монохромного адаптера;
(ES:DI) - указывает на таблицу:
слово 1 - номер модели дисплея;
2 - число PEL по вертикали (на метр);
3 - число PEL по горизонтали (на метр);
4 - число PEL по вертикали,
5 - число PEL по горизонтали,
6 - расстояние между PEL по горизонтали (мм)
7 - расстояние между PEL по вертикали (мм).

В PC Convertible определены типы дисплеев, перечисленные в табл. 7.

Слово	Монохромный дисплей	CGA	VGA как CGA	VGA как монохромный
1	5151H	5153H	5140H	5140H
2	0	0498H	08E1H	0
3	0	OA15H	0987H	0
4	0	00C8H	00C8H	0
5	0	0280H	0280H	0
6	0	0352H	01B8H	0
7	0	0184H	019AH	0

Типы дисплеев PC Convertible.

23. Получить или установить код типа дисплея

Две следующие функции поддерживаются только в VGA.

1. Получить код типа дисплея

Параметры: (AX) = 1A00H.
Результат: (AL) = 1AH как признак того, что функция поддерживается,
(BL) - код активного дисплея,
(BH) - код альтернативного дисплея.

Следующие коды дисплеев определены для VGA:
0 - нет дисплея,
1 - монохромный адаптер с5151,
2 - CGA с 5153/4,
4 - EGA с 5153/4,

5 - EGA с 5151,
6 - система профессиональной графики с 5175
7 - аналог адаптера VGA с аналогичным монохромным дисплеем;
8 - аналог адаптера VGA с аналогичным цветовым дисплеем;
11 - аналог адаптера VGA с аналогичным монохромным дисплеем;
12 - аналог адаптера VGA с аналогичным цветовым дисплеем;
255 - неизвестный адаптер;
остальные коды резервируются.

2. Установить код типа дисплея

Параметры: (AX) = 1A01H.
(BL) - код активного дисплея,
(BH) - код альтернативного дисплея.
Результат: (AL) = 1AH как признак того, что функция поддерживается.

24. Получить информацию о состоянии

Следующая функция поддерживается только в VGA.
Параметры: (AH) = 1BH,
(BX) = 0,
(ES:DI) - указатель буфера для результатов (40H байтов);
Результат: (AL) = 1BH как признак того, что функция поддерживается.

Буфер (ES:DI) заполняется следующим образом:
OOH (слово) - смещение области статической информации;
02H (слово) - сегмент области статической информации.

Следующие поля устанавливаются динамически и отражают текущий режим дисплея:

04H (байт) - режим дисплея (см. функцию (AH) = 0);
05H (слово) - число столбцов текстового режима;
07H (слово) - длина буфера регенерации (байтов);
09H (слово) - начальный адрес буфера регенерации;
OBH (слова) - позиции курсора для восьми страниц (строка, столбец);
1BH (слово) - тип (размер) курсора;
1DH (байт) - номер активной страницы;
1EH (слово) - базовый адрес портов адаптера (3Bx - MDA, 3Dx - CGA);
20H (байт) - текущая установка регистра 3x8;
21H (байт) - текущая установка регистра 3x9;
22H (байт) - число текстовых строк на экране;
23H (слово) - число скан-линий на знак;
25H (байт) - код активного дисплея;

26Н (байт) - код альтернативного дисплея;
27Н (слово) - палитра, поддерживаемая для текущего режима дисплея;
29Н (слово) - страницы, поддерживаемые для текущего режима дисплея;
2АН (байт) - число скан-линий для активного режима:
2ВН (байт) - блок первичного знака (см. функцию (АХ) = 1103Н);
2СН (байт) - блок вторичного знака (см. функцию (АХ) = 1103Н);
2DH (байт) - разное:
биты 7, 6 - резерв;
бит 5 = 0 - интенсивность фона, 1 - мерцание;
бит 4 = 1 - активность эмуляции курсора (в модели 30 - всегда 0);
бит 3 = 1 - запрещен режим установки умолчания для палитры;
бит 2 = 1 - подключен монохромный дисплей;
бит 1 = 1 - активно суммирование цветовых интенсивностей;
бит 0 = 1 - активны все режимы дисплея (в модели 30 - всегда 0);
2ЕН (байты) - резерв;
31Н (байт) - объем доступной памяти в буфере дисплея:
О - 64К, 1 - 128К, 2 - 192К, 3 - 256К, остальное - резерв;
32Н (байт) - сохраненная информация:
биты 7, 6 - резерв;
бит 5 = 1 - активно расширение DCC,
бит 4 = 1 - активно переопределение палитры,
бит 3 = 1 - активно переопределение графических фонтов;
бит 2 = 1 - активно переопределение текстовых фонтов;
бит 1 = 1 - активна динамическая область сохранения;
бит 0 = 1 - активен 512-байтовый набор знаков.
33Н (13 байт) - резерв.

Область статической информации длиной 16 байтов заполнена следующим образом (нулевое значение некоторого бита означает, что соответствующее средство не поддерживается):
ООН - режимы дисплея:
биты 7 - 0 - режимы 7 - 0;
01Н - режимы дисплея:
биты 7 - 0 - режимы 15 - 8;
02Н - режимы дисплея:
биты 3 - 0 - режимы 19 - 16,
биты 7 - 4 - резерв;
О3Н - 06Н - резерв;
07Н - число скан-линий в текстовых режимах:
бит 0 - 200,
бит 1 - 350,
бит 2 - 400,
биты 3 - 7 - резерв;
08Н - блоки знака доступны в текстовых режимах

09H - максимальное число активных блоков знаков в текстовых режимах
(см. функцию (AH) = 11H);
OAH - разное:
бит 7 - страницы палитры (см. (AH) = ЮН; для модели 30 - всегда 0),
бит 6 - палитра (см. (AH) = ЮН),
бит 5 - палитра EGA (см. (AH) = ЮН),
бит 4 - эмуляция курсора (см. (AH) = 1),
бит 3 - режим загрузки умолчания для палитры (см. (AH) = 12H),
бит 2 - загрузка фонтов (см. (AH) = 11H),
бит 1 - суммирование цветовых интенсивностей (см. (AH) = ЮН и 12H),
бит 0 - все режимы всех дисплеев (в модели 30 всегда 0);
OBH - разное:
биты 7 - 4 - резерв,
бит 3 - DCC (см. (AH) = 1AH),
бит 2 - управление интенсивностью и мерцанием фона (см. (AH) = ЮН),
бит 1 - сохранение и восстановление (см. (AH) = 1CH, всегда 0),
бит 0 - световое перо (см. (AH) = 4);
0CH - 0DH - резерв;
OEH - функции сохранения:
биты 7, 6 - резерв;
бит 5 = 1 - расширение DCC (в модели 30 всегда 0),
бит 4 = 1 - переопределение палитры,
бит 3 = 1 - переопределение графических фонтов,
бит 2 = 1 - переопределение текстовых фонтов,
бит 1 = 1 - динамическая область сохранения,
бит 0 = 1 - 512-байтовый набор знаков;
OFH - резерв.

25. Сохранение и восстановление состояния дисплея.

Следующие функции поддерживаются только в VGA.

1. Получить размер буфера для сохранения/восстановления.

Параметры: (AX) = 1C00H,
(CX) - код назначения буфера (см. ниже).
Результат: (AL) = 1CH как признак того, что функция поддерживается,
(BX) - размер буфера (в единицах по 64 байта).

2. Сохранить состояние.

Параметры: (AX) = 1C01H,
(CX) - код назначения (см. ниже),
(ES:BX) - указатель буфера для сохранения.
Результат: (AL) = 1CH как признак того, что функция поддерживается.

3. Запрос сохраненного состояния.

Параметры: (AX) = 1C02H,
(CX) - код назначения (см. ниже),
(ES:BX) - указатель буфера.
Результат: (AL) = 1CH как признак того, что функция поддерживается.
Сохраненное состояние восстановлено.
Код назначения (единичное значение бита указывает
на сохранение/восстановление соответствующих средств):
биты 15 - 3 резервируются и должны быть нулевыми,
бит 2 - динамическая область состояния (DAC) и регистры цвета,
бит 1 - область данных BIOS,
бит 0 - аппаратное состояние.

Замечание: при сохранении состояния оно изменяется; для продолжения работы требуется выполнить его восстановление.

Определение набора подключенного оборудования (прерывание 11H).

Прерывание 11H возвращает в регистре AX информацию об оборудовании компьютера. Ту же информацию можно получить, прочитав слово с адресом 0000:0410H. Биты слова конфигурации оборудования кодируются следующим образом:

биты 15 - 14 - число подключенных устройств печати (0 - 3); бит 13 (в некоторых моделях) - установлен внутренний модем; бит 12 (в некоторых моделях) - установлен адаптер игр; биты 11 - 9 - число адаптеров или портов данных асинхронной связи;

бит 8 - не используется;

биты 7 - 6 (только если бит 0 равен 1) - число дискетных устройств (00 - 1, 01 - 2, 10 - 3, 11 - 4);

биты 5 - 4 - режим экрана, устанавливаемый при инициализации BIOS:

00 - резервируется,
01 - режим 1(40*25, см. 11.1),
10 - режим 3 (80*25),
11 - режим 7 (80*25);

биты 3 - 2 - резервируются (в некоторых моделях указывают объем памяти на системной плате);

бит 2 - установлено устройство типа "мышь";

бит 1 - установлен арифметический сопроцессор;

бит 0 - имеется хотя бы одно дискетное устройство, с которого возможна загрузка системы.

Единичное значение бита указывает на установленное оборудование.

Замечания: для определения числа установленных жестких дисков можно использовать вызов функции 8 INT 13H.

Наличие сопроцессора распознается только по установке соответствующего переключателя на системной плате. Лучший способ проверить наличие сопроцессора - попытаться выполнить его инструкцию и проверить результат. Использование прерывания 11Hдля определения подключенного оборудования - устаревший подход.

Определение объема памяти (прерывание 12H).

Прерывание 12H возвращает в регистре AX объем памяти в килобайтах. Ту же информацию можно получить, прочитав слово с адресом 0000:0413H. Независимо от модели компьютера возвращается только объем системной памяти (до адреса A000:0000); значение не может превосходить 640K. При определении объема системной памяти BIOS проверяет ее непрерывность и пригодность, но начальный объем для проверки во многих компьютерах определяется установкой переключателей конфигурации на системной плате.

Для определения объема дополнительной памяти служат функции 88H и OC1H прерывания 15H.

Работа с асинхронным адаптером (прерывание 14H).

Следующие функции INT 14H поддерживают протокол RS-232-C:

(AH) = 0 - инициализация порта асинхронной связи;
1 - передача байта;
2 - прием байта;
3 - получить состояние;
4 - расширенная инициализация;
5 - расширенное управление портом связи.

Сохраняются значения всех регистров, кроме AX. Базовые адреса портов адаптеров связи и значения тайм-аутов устанавливаются при инициализации BIOS.

1. Инициализация порта асинхронной связи.

Параметры: (AH) = 0;
(DX) - номер канала (0 - 3) в соответствии с базовым адресом портов в таблице BIOS по адресу 40:0;
(AL) - параметры инициализации:
биты 1,0 = 00 - 5-ти битный код,
= 01 - 6-ти битный код,
= 10 - 7-ми битный код,
= 11 - 8-ми битный код;
бит 2 = 0 - 1 стоп-бит,
1 - 2 стоп-бита для 6-8-ми битного кода,
1,5 стоп-бита для 5-ти битного кода;
бит 3 = 0 - нет контроля по паритету,
= 1 - есть контроль по паритету;

бит 4 = **0** - контроль по нечетности,
= 1 - контроль по четности;
биты 7-5 - скорость обмена (бод):
= **000 - 110**
= **100 - 150**
= 010 - 300
= 110-600
= 001 - 1200
= 101 - 2400
= 011-4800
= **111 - 9600**

Результаты: (АН) - состояние линии **управления,**
(AL) - состояние модема (см. **15.4).**

2. Передача байта

Параметры: **(АН) = 1;**
(DX) - номер канала (0 - 3) в соответствии с базовым адресом портов в таблице BIOS по адресу 40:0;
(AL) - байт данных для передачи.

Результаты: (АН) - состояние линии управления (см. **15.4),**
(AL) - сохраняется.

Если бит 7 состояния линии устанавливается в 1, то остальные биты состояния непредсказуемы. **Байт** в этом случае не передан.

3. Прием байта

Параметры: **(АН)** = 2;
(DX) - номер канала (0 - 3) в соответствии с базовым адресом портов в таблице BIOS по адресу 40:0;
(AL) - байт данных для передачи.

Результаты: (АН) - состояние линии управления (см. **15.4),**
(AL) - принятый байт данных.

Модуль BIOS ожидает приема байта.

4. Получить состояние канала.

Параметры: (АН) = 3;
(DX) - номер канала (0 - 3) в соответствии с базовым адресом портов в таблице BIOS по адресу 40:0;

Результаты: **(АН) - состояние линии** управления:
бит 7 - тайм-аут
6 - конец передачи (КПД)
5 - готов к передаче (ГПД)

4 - обрыв канала (авария)
3 - ошибка по стоп-биту
2 - ошибка по паритету
1 - переполнение
0 - готов к приему (ГПР)
(AL) - состояние модема:
бит 7 - детектор принимаемого линейного сигнала канала данных (цепь 109),
6 - индикатор вызова (цепь 125),
5 - аппаратура готова (цепь 107),
4 - готов к передаче (цепь 106).

5. Расширенная инициализация

Вызов следующей функции используется для инициализации портов асинхронной связи.

Параметры: (AH) = 4;
(DX) - номер канала (0 - 3) в соответствии с базовым адресом портов в таблице BIOS по адресу 40:0;
(AL) = 0 - без обработки Break,
1 - с обработкой Break,
(BH) - контроль паритета:
0 - нет контроля,
1 - контроль по нечетности,
2 - контроль по четности,
3 - stick (?) контроль по нечетности,
4 - stick (?) контроль по четности,
(BL) = 0 - 1 стоп-бит,
1 - 2 стоп-бита для 6-8-ми битного кода,
1,5 стоп-бита для 5-ти битного кода;
(CH) = 0 - 5-ти битный код,
1 - 6-ти битный код,
2 - 7-ми битный код,
3 - 8-ми битный код;
(CL) - скорость обмена (бод):
0 - 110
1 - 150
2 - 300
3 - 600
4 - 1200
5 - 2400
6 - 4800
7 - 9600
8 - 19200.

Результаты: (AH) - состояние линии управления,
(AL) - состояние модема (см. 4).

6. Расширенное управление портом связи

Две следующие функции применяются для управления модемом:

(AX) = 0500H - чтение регистра управления модемом,

(AX) = 0501H - запись в регистр управления модемом.

1. Параметры: (AX) = 0500H,
(DX) - номер канала (0 - 3) в соответствии с базовым адресом портов в таблице BIOS по адресу 40:0;

Результат: (BL) - состояние регистра управления модемом:
бит 0 - терминал данных готов (DTR, цепь 108),
бит 1 - запрос на передачу (цепь 105),
бит 2 - управление выходом OUT1,
бит 3 - управление выходом OUT2,
бит 4 - цикл (возможность диагностирования),
биты 5 - 7 - нули.

2. Параметры: (AX) = 0501H,
(DX) - номер канала (0 - 3) в соответствии с базовым адресом портов в таблице BIOS по адресу 40:0;
(BL) - байт для записи в регистр управления модемом (биты как в 1).

Результаты: (AH) - состояние линии управления,
(AL) - состояние модема (см. 15.4).

Работа с гибкими и жесткими дисками (прерывание 13H).

BIOS предоставляет ряд функций для работы с дискетами и с жесткими дисками. Неверное их использование может разрушить всю информацию на носителе. Как правило, для обслуживания дисков достаточно средств DOS, использование INT 13H может потребоваться лишь для нестандартных носителей или для обработки дискет, защищенных от копирования.

Дисковые функции BIOS перечислены в табл. 8. Только функции 0 - 5 применимы ко всем дискам независимо от установленного контроллера и типа носителя. Как правило, в регистре (DL) указывается номер устройства, при этом номера 0 - 7FH отсылают к дискетным устройствам, номера 80H - OFFH - к жестким дискам. Корректность номера устройства проверяется при вызове каждой функции.

В некоторых моделях компьютеров перед выполнением любой дискетной функции, требующей включения двигателя, вызывается функция (AX) = 90FDH прерывания 15H, информирующая систему, чтобы она могла выполнять другие задачи во время разгона двигателя. Двигатели жестких дисков всегда включены.

В некоторых моделях компьютеров при выполнении INT 13H BIOS вызывает функцию 90H прерывания 15H перед тем, как перейти к ожиданию аппаратного пре-

рывания от дискового устройства; вызов сообщает операционной системе, что она должна ждать освобождения устройства. После обработки аппаратного прерывания от диска вызывается функция 91H прерывания 15H, чтобы сообщить о завершении дисковой операции. Вызовы функций 90H и 91H передают системе тип диска (дискета, жесткий диск).

номер	функция
0	сброс дисков
1	получить состояние
2	чтение секторов
3	запись секторов
4	проверка секторов
5	разметка дорожки
6	разметка дефектной дорожки
7	разметка диска
8	получить параметры устройства
9	установить параметры устройства
10	резервируется для диагностики
11	резервируется для диагностики
12	установка
13	альтернативный сброс диска
14	резервируется для диагностики
15	резервируется для диагностики
16	проверка готовности устройства
17	вывод диска на дорожку 0
18	резервируется для диагностики
19	резервируется для диагностики
20	резервируется для диагностики
21	получить тип диска
22	состояние линии замены дискеты
23	установить тип диска для разметки
24	установить тип носителя для разметки

Функции BIOS для поддержки дисков

1. Сброс дисков

Параметры: (AH) = 0,
(AL) - номер устройства.
Результат: (CF) = 0 - нормальное завершение,
= 1 - ошибка, и в AH ее код (см. табл. 7).

Если (DL) < 80H, то сбрасываются только дискетные устройства, иначе - все диски. В последнем случае в (AH) при ошибке возвращается состояние жесткого диска; чтобы получить состояние дискеты, нужно прочитать байт с абсолютным адресом 0000:0441H.

Выполнение сброса состоит в сбросе контроллера, сбросе устройства, выбранного контроллером (устройства, к которому было последнее обращение), и в установке носителя на начальную дорожку. Сброс обычно вызывается после ошибок выполнения других функций.

Замечание: жесткие диски будут сбрасываться вызовом данной функции лишь в том случае, если младшие 7 битов (DL) указывают номер жесткого диска, фактически установленного в компьютере.

2. Получить состояние

Параметры: (AH) = 1,
(DL) - номер устройства.
Результат: (AH) - состояние завершения последней дисковой операции,
(CF) = 0 - состояние 0 (см. табл. 9)
= 1 - иначе.

На результат оказывает влияние только старший бит (DL), определяющий тип диска, но корректность номера устройства проверяется.

Коды завершения дисковых функций BIOS перечислены в табл. 9.

код	состояние	тип диска
00	ошибки не распознаны	любой
01	неверный код функции	любой
02	не найден маркер адреса	любой
03	попытка нарушения защиты записи	любой
04	сектор не найден	любой
05	ошибка сброса	жесткий

06	активность линии замены дискеты	дискета
07	ошибка установки параметров диска	жесткий
08	НДП не справляется с обменом	любой
09	попытка выйти за физический сегмент при обмене через КНДП	любой
OA	распознан флаг дефектного сектора	жесткий
0B	распознан неверный номер цилиндра	жесткий
OC	тип дискеты не распознан	дискета
OD	неверное число секторов при разметке	жесткий
OE	распознан маркер управляющих данных	жесткий
OF	ошибка уровня управления КНДП	жесткий
10	ошибка контрольной суммы данных	любой
11	исправленная ошибка контрольной суммы данных	жесткий
20	общая неисправность контроллера	любой
40	ошибка поиска	любой
80	устройство не готово	любой
BB	неопределенная ошибка	жесткий
CC	ошибка записи	жесткий
EO	ошибка состояния контроллера	жесткий
FF	ошибка получения состояния	жесткий

Коды завершения дисковых функций

Замечание: состояние последней дискетной операции хранится в байте памяти с адресом 0000:0441H, состояние последней операции с жестким диском - в байте 0000:0474H.

3. Чтение секторов

Параметры: (AH) = 2,
(AL) - число секторов для чтения,
(DL) - номер устройства,
(DH) - номер головки,
(CH) - номер дорожки,
(CL) - номер сектора,
(ES:BX) - адрес буфера.

Результат: (CF) = 0 - нормальное выполнение,
= 1 - ошибка,
(AH) - код завершения,
(AL) - число фактически прочитанных секторов.

Операция считывает данные указанного числа секторов в память. Значения параметров, за исключением номера устройства, не проверяются. Для дискетных устройств все запрашиваемые секторы должны располагаться на одной физической дорожке дискеты, для жесткого диска такого ограничения нет.

При обращении к жесткому диску (CH) содержит 8 младших бит номера дорожки, два старших бита размещаются в битах 7 и 6 (CL).

Замечание: указание (AL) = 0 приведет к чтению некоторого числа секторов, определяемого контроллером.

4. Запись секторов

Параметры: (AH) = 3,
(AL) - число секторов для записи,
(DL) - номер устройства,
(DH) - номер головки,
(CH) - номер дорожки,
(CL) - номер сектора,
(ES:BX) - адрес буфера.
Результат: (CF) = 0 - нормальное выполнение,
= 1 - ошибка,
(AH) - код завершения,
(AL) - число фактически записанных секторов.

Операция записывает на носитель данные указанного числа секторов. Значения параметров, за исключением номера устройства, не проверяются. Для дискетных устройств все запрашиваемые секторы должны располагаться на одной физической дорожке дискеты, для жесткого диска такого ограничения нет.

При обращении к жесткому диску (CH) содержит 8 младших бит номера дорожки, два старших бита размещаются в битах 7 и 6 (CL).

Замечание: указание (AL) = 0 приведет к записи некоторого числа секторов, определяемого контроллером.

5. Проверка секторов

Параметры: (AH) = 4,
(AL) - число секторов для проверки,
(DL) - номер устройства,
(DH) - номер головки,
(CH) - номер дорожки,

(CL) - номер сектора,
(ES:BX) - адрес буфера
Результат: (CF) = 0 - нормальное выполнение,
= 1 - ошибка,
(AH) - код завершения,
(AL) - число фактически обработанных секторов.

Операция проверяет данные указанного числа секторов, выполняя контрольное чтение. Данные не поступают в память компьютера. Значения параметров, за исключением **номера устройства, не проверяются. Для дискетных устройств все запрашиваемые секторы должны располагаться на одной физической дорожке дискеты, для жесткого диска такого ограничения** нет.

При обращении к жесткому диску (CH) содержит 8 младших бит номера дорожки, два старших бита размещаются в битах 7 и 6 (CL).

Замечания:
1. Указание (AL) = 0 приведет к проверке некоторого числа секторов, определяемого контроллером.
2. ES:BX не требуется для многих контроллеров (AT, PC Convertible, System/2).

6. Разметка дорожки

Параметры: (AH) = 5,
(AL) - число секторов на дорожке дискеты или
коэффициент чередования для жесткого диска,
(CH) - номер цилиндра (для жесткого диска - 8 младших бит),
(CL) - для жесткого диска в битах 6 - 7 старшие биты
номера цилиндра,
(DL) - номер устройства,
(DH) - номер головки,
(ES:BX) - указатель буфера, содержащего информацию
для разметки (зависит от контроллера, см. ниже).
Результат: (CF) = 0 - нормальное выполнение,
(CF) = 1 - ошибка, и в AH ее код.

Функция размечает одну дорожку дискеты или жесткого диска. Параметры, за исключением номера устройства, не проверяются.

Для дискеты ES:BX указывает на блок данных, содержащий четырехбайтовую запись для каждого сектора, создаваемого на дорожке:
байт 0 - номер цилиндра,
байт 1 - номер головки,
байт 2 - номер сектора,
байт 3 - код размера сектора:
0 - 128 байт,
1 - 256 байт,
2 - 512 байт,
3 - 1024 байта.

Для устройств, поддерживающих более одного формата дискеты, перед разметкой необходимо вызвать функцию 17H или 18H прерывания 13H, иначе при разметке будут использоваться максимальные параметры носителя, поддерживаемые устройством. Некоторые компьютеры не поддерживают этих функций; в этом случае перед разметкой необходимо прямо изменить таблицу параметров BIOS дискеты. Эта таблица адресуется вектором прерывания 1EH. После разметки необходимо восстановить значения параметров таблицы.

Некоторые контроллеры (в частности, большинство контроллеров жестких дисков) не позволяют произвольно нумеровать и располагать на дорожке секторы диска. Для таких контроллеров в (AL) указывается коэффициент чередования, определяющий физический порядок секторов на дорожке диска. Поле данных, адресуемое ES:BX, в таких случаях либо не используется, либо для жестких дисков содержит другую информацию (см. ниже).

Хотя контроллеры нумеруют секторы дискеты, начиная с 0, в BIOS используется нумерация с 1, так что стандартное поле для разметки девятисекторной дискеты DOS имеет вид:

```
0 1 1 2 0 1 2 2 ... 0 1 9 2
```

Для жестких дисков ES:BX указывает на 512-байтовое поле данных, первые байты которого содержат информацию о разметке, а остальные игнорируются. Для каждого сектора диска (в том порядке, в каком они будут физически расположены на дорожке) указывается:

байт 0 = 00 - разметить как обычный сектор,
80H - разметить с флагом дефектного сектора;
байт 1 - номер сектора.

Некоторые контроллеры не пользуются этой информацией, используя только коэффициент чередования в (AL) (см. также функцию 7).

Замечание: в некоторых моделях компьютеров функции разметки жестокого диска не поддерживаются BIOS; в этом случае вызов данной функции вернет код завершения 01 (неверная функция).

7. Разметка дефектной дорожки

Параметры: (AH) = 6,
(AL) - коэффициент чередования для жесткого диска,
(CH) - номер цилиндра (8 младших бит),
(CL) - в битах 6 - 7 старшие биты номера цилиндра,
(DL) - номер устройства,
(DH) - номер головки,
Результат: (CF) = 0 - нормальное выполнение,
(CF) = 1 - ошибка, и в AH ее код.

Функция размечает одну дорожку жесткого диска, отмечая все ее секторы как дефектные. Параметры, за исключением номера устройства, не проверяются.

Замечание: функция не поддерживается некоторыми контроллерами и BIOS некоторых компьютеров (см. функцию 5). Функция неприменима к дискетам.

8. Разметка диска, начиная с указанной дорожки.

Параметры: (AH) = 7,
(AL) - коэффициент чередования для жесткого диска,
(CH) - начальный номер цилиндра (8 младших бит),
(CL) - в битах 6 - 7 старшие биты номера цилиндра,
(DL) - номер устройства,
(DH) - начальный номер головки,
Результат: (CF) = 0 - нормальное выполнение,
(CF) = 1 - ошибка, и в AH ее код.

Функция размечает жесткий диск, начиная с указанной дорожки. Параметры, за исключением номера устройства, не проверяются.

Замечание: функция не поддерживается многими контроллерами и BIOS некоторых компьютеров (см. функцию 5), в частности, в IBM AT и в System/2 вырабатывается код ошибки 01. Функция неприменима к дискетам.

9. Получить параметры устройства

Параметры: (AH) = 8,
(DL) - номер устройства.
Результат: (CF) = 0 - нормальное выполнение,
(ES:DI) - указатель на 11-байтовый блок параметров дискеты (см. табл. 8), не используется для жестких дисков,
(CH) - максимальный номер цилиндра (младшие 8 бит),
(CL) - максимальное число секторов на дорожке в битах 5 - 0, старшие биты максимального номера цилиндра в битах 6 - 7,
(DH) - максимальный номер головки,
(DL) - число жестких дисков, поддерживаемых контроллером, или общее число дискетных устройств.
(BH) = 0 (только для дискет),
(BL) - тип дискетного устройства в битах 3 - 0:
001 - 360 Кбайт, 40 дорожек на стороне мини-дискеты,
010 - 1.2 Мбайта, 80 дорожек на стороне мини-дискеты,
011 - 720 Кбайт, 80 дорожек на стороне микродиска,
100 - 1.44 Мбайта, 80 дорожек на стороне микродиска.

Для дискетных устройств функция поддерживается только некоторыми компьютерами. При этом тип устройства в (BL) возвращается только в том случае, если он известен компьютеру (например, при использовании CMOS или специальных переключателей конфигурации); в прочих случаях возвращается (BL) = 0.

Вызов функции 8 возвращает максимальные параметры дискетных устройств независимо от установленного носителя. Для жестких дисков возвращаются параметры установленных дисков (они должны быть одинаковыми для всех дисков, подключенных к контроллеру). Структура таблицы параметров дискеты приведена в п. 24, структура параметров жесткого диска - в п. 27.

10. Установка параметров жесткого диска

Параметры: (AH) = 9,
(DL) - номер устройства.
Результат: (CF) = 0 - нормальное выполнение,
(CF) = 1 - ошибка, и в AH ее код.

Функция поддерживается всеми компьютерами, но ее выполнение зависит от модели, и прежде всего от установленного контроллера жесткого диска. Для большинства моделей параметры выбираются из таблицы параметров жестких дисков в ПЗУ соответственно установке переключателей на плате адаптера. Таблица параметров адресуется вектором 41H и после загрузки системы может быть переопределена. Параметры жесткого диска перечислены в табл. 11. моделях (IBM AT, System/2), допускающих присоединение к одному контроллеру двух дисков с разными параметрами, параметры устройства 0 определяются вектором 41H, параметры устройства 1 - вектором 46H. Наконец, для некоторых контроллеров эта функция не вызывает никаких действий: контроллер получает параметры непосредственно от устройства.

11. Установка жесткого диска

Параметры: (AH) = 0CH,
(DL) - номер устройства,
(DH) - номер головки,
(CH) - младшие биты номера цилиндра,
(CL) - старшие биты номера цилиндра в битах 6-7.
Результат: (CF) = 0 - нормальное выполнение,
(CF) = 1 - ошибка, и в AH ее код.

Выполняется установка на указанную дорожку и выбор головки. Функция может использоваться для увеличения производительности системы, если BIOS и контроллер допускают совмещение операций установки на разных дисках.

12. Альтернативный сброс диска

Параметры: (AH) = 0DH,
(DL) - номер устройства.
Результат: (CF) = 0 - нормальное выполнение,
(CF) = 1 - ошибка, и в AH ее код.

Функция применима только к жестким дискам и вызывает сброс контроллера и указанного жесткого диска (см. 14.1).

13. Проверка готовности устройства

Параметры: (AH) = 10H,
(DL) - номер устройства.
Результат: (CF) = 0 - нормальное выполнение,
(CF) = 1 - ошибка, и в AH ее код.

Функция применима только к жестким дискам.

Для некоторых контроллеров выполнение этой функции вызывает установку диска на дорожку 0.

14. Установка диска на дорожку О

Параметры: (AH) = 10H,
(DL) - номер устройства.
Результат: (CF) = 0 - нормальное выполнение,
(CF) = 1 - ошибка, и в AH ее код.

Функция применима только к жестким дискам.

15. Получить тип диска

Параметры: (AH) = 15H,
(DL) - номер устройства.
Результат: (CF) = 0 - нормальное выполнение,
(AH) - тип диска:
0 - диск не установлен,
1 - дискета, линия замены не поддерживается,
2 - дискета, линия замены поддерживается,
3 - жесткий диск;
(CX:DX) - общее число секторов на диске,
(CF) = 1 - ошибка, и в AH ее код.

Функция была введена для поддержки контроллеров, допускающих присоединение и дискетных устройств, и жестких дисков, и не поддерживается многими компьютерами.

16. Получить состояние линии замены дискеты

Параметры: (AH) = 16H,
(DL) - номер устройства.
Результат: (CF) - флаг завершения,

(AH) - код возврата:
OOH - линия замены дискеты неактивна,
01H - неверный номер устройства,
06H - линия замены дискеты активна,
80H - устройство не готово.

Функция применима только к дискетам и не поддерживается многими компьютерами. Флаг (CF) устанавливается обычным образом, но (CF) = 1 не свидетельствует об ошибке при (AH) = 6.

17. Установить тип устройства для разметки

Параметры: (AH) = 17H,
(DL) - номер устройства,
(AL) - тип дискеты и устройства:
1 - дискета 320/360 Кбайт на обычном устройстве,
2 - дискета 360 Кбайт на устройстве большой емкости,
3 - дискета 1.2 Мбайта на устройстве большой емкости,
4 - дискета 720 Кбайт на соответствующем устройстве.
Результат: (CF) = 0 - нормальное выполнение,
(CF) = 1 - ошибка, и в AH ее код.

Функция применима только к дискетам и не поддерживается многими компьютерами. Ее использование необходимо перед разметкой дискеты на устройстве, поддерживающем различные форматы дискет. Код должен соответствовать фактическим возможностям устройства.

18. Установить тип носителя для разметки

Параметры: (AH) = 18H,
(DL) - номер устройства,
(CH) - число дорожек (возможно, младшие 8 бит),
(CL) - число секторов на дорожке и, возможно,
два старших бита числа дорожек.
Результат: (CF) = 0 - нормальное выполнение,
(ES:DI) - указатель на таблицу параметров дискетного устройства,
(CF) = 1 - ошибка, и в AH ее код.

Функция применима только к дискетам и не поддерживается многими компьютерами. Ее использование необходимо перед разметкой дискеты на устройстве, поддерживающем различные форматы дискет. Для каждого типа носителя поддерживается только один блок параметров устройства (см. табл.).

Для устройств с линией замены дискеты эта функция прежде всего пытается сбросить активность линии (если она активна), что соответствует установке новой диске-

ты. Если это не удается, возвращается ошибка. Таким образом, при отсутствии дискеты в устройстве возвращается ошибка.

19. Парковать головки

Данная функция поддерживается только в System/2.

Параметры: (AH) = 19H,
(DL) - номер устройства;
Результат: (CF) = 0 - нормальное выполнение,
(CF) = 1 - ошибка, и в AH ее код.

Функция полезна для защиты жесткого диска перед выключением питания.

20. Разметка устройства

Функция применима только в системе с контроллером IBM ESDI жесткого диска.

Параметры: (AH) = 1AH,
(AL) = 0 - с данным запросом не связано никакой таблицы адресов дефектных блоков;
иначе - счетчик таблицы дефектных блоков.
(ES:BX) - адрес таблицы дефектных блоков;
(CL) - модификаторы:
биты 7 - 5 должны быть нулями;
бит 4 - периодические прерывания. Контроллер прерывает систему после завершения каждой из трех фаз разметки каждого цилиндра, чтобы позволить системе управлять процессом разметки. Фазы разметки: 0 - резерв, 1 - анализ поверхностей, 2 - форматирование. Код фазы доступен BIOS через прерывание 15Hc (AH) = OFH. При возврате очищенный флаг CF приводит к продолжению разметки, установленный - прекращает разметку.
бит 3 - расширенный анализ поверхностей. Прежде чем размечать дорожку функцией с единичным значением этого бита, необходимо попытаться разметить ее функцией с нулевым значением бита.
бит 2 - обновление карты вторичных дефектов. Если этот бит установлен в 1, то составляется новая карта вторичных дефектов.
бит 1 - игнорирование карты вторичных дефектов;
бит 0 - игнорирование карты первичных дефектов.
(DL) - номер устройства.

Расширенный сервис AT (прерывание 15).

В первых версиях BIOS прерывание **15H** использовалось для поддержки кассетных накопителей на магнитной ленте. Эти устройства не получили широкого распространения в ПП ЭВМ и во многих моделях соответствующие средства (функции **0 - 3**) не поддерживаются. В более поздних версиях BIOS через INT **15H** вызываются разнообразные средства поддержки операционных систем. Список функций INT **15H** приведен в следующей таблице:

код	Функция
00	включение двигателя кассетного устройства
01	выключение двигателя кассетного устройства
02	чтение блоков с кассеты
03	запись блоков на кассету
OF	периодические прерывания разметки диска
21	Установить область для фиксации ошибок теста при включении питания
40	чтение и модификация системных параметров
41	ожидание внешнего события
42	запрос сброса системы
43	чтение состояния системы
44	включение или выключение внутреннего модема
4F	перехват клавиатурных кодов
80	открыть устройство
81	закрыть устройство
82	прекратить программу устройства
83	ожидание события
84	поддержка координатных ручек
85	поддержка клавиши SysReq
86	задержка
87	перемещение блока памяти
88	определение объема дополнительной памяти
89	переключение процессора в привилегированный режим
90	сообщение о занятости устройства
91	сообщение о готовности устройства

C0	получить параметры модели компьютера
C1	получить сегмент расширения области данных BIOS
C2	интерфейс с устройствами типа «мышь»
C3	разрешение или запрещение «сторожевого пса»
C4	выбор программируемых режимов

1. Включение двигателя кассетного устройства

Параметры: (AH) = 0.
Результат: (CF) = 0 и (AH) = 0.

В большинстве моделей компьютеров возвращается (CF) = 1 как признак того, что кассетная лента не поддерживается. В некоторых моделях при неисправности или отсутствии накопителя возвращается (AH) = 86H.

2. Выключение двигателя кассетного устройства

Параметры: (AH) = 1.
Результат: (CF) = 0 и (AH) = 0.

В большинстве моделей компьютеров возвращается (CF) = 1 как признак того, что кассетная лента не поддерживается. В некоторых моделях при неисправности или отсутствии накопителя возвращается (AH) = 86H.

3. Чтение блоков с кассеты

Параметры: (AH) = 2,
(ES:BX) - указатель буфера,
(CX) - счетчик байтов.
Результат: (CF) = 0 - нормальное выполнение,
(ES:BX) - указатель на первый свободный байт буфера,
(DX) - число фактически прочитанных байтов
или (CF) = 1 - флаг ошибки,
(AH) - код ошибки: 01 - ошибка контрольной суммы данных,
02 - потеря данных при чтении,
04 - данные не найдены.

Код ошибки в AH возвращается только в PC-jr.

В большинстве моделей компьютеров возвращается (CF) = 1 как признак того, что кассетная лента не поддерживается. В некоторых моделях при неисправности или отсутствии накопителя возвращается (AH) = 86H.

4. Запись блоков на кассету

Параметры: (AH) = 3,
(ES:BX) - указатель буфера,
(CX) - счетчик байтов.
Результат: (CF) = 0 - нормальное выполнение,
(ES:BX) - указатель байт буфера, следующий за последним записанным байтом,
(DX) - число фактически записанных байтов
или (CF) = 1 - флаг ошибки,
(AH) - код ошибки: 01 - ошибка контрольной суммы данных,
02 - потеря данных при записи,
04 - данные не найдены.

Код ошибки в AH возвращается только в PC-jr.

В большинстве моделей компьютеров возвращается (CF) = 1 как признак того, что кассетная лента не поддерживается. В некоторых моделях, при неисправности или отсутствии накопителя возвращается (AH) = 86H.

5. Периодические прерывания разметки диска.

Следующая функция используется только при работе с адаптером IBM ESDI жесткого диска. Прерывание 15H с (AH) = OFH может вызываться при выполнении разметки каждого цилиндра диска (см. 14.20) трижды: (AL) = 0 - перед разметкой цилиндра, (AL) = 1 - после анализа поверхностей, (AL) = 2 - после форматирования.

BIOS устанавливает (CF) = 1, что приводит к прекращению разметки. Чтобы разрешить разметку жесткого диска контроллеру ESDI, нужно по крайней мере заменить обработчик данной функции на модуль, возвращающий (CF) = 0.

Параметры: (AH) = OFH,
(AL) - код фазы (0 - 2),
Результат: (CF) = 0 - разметка будет продолжена,
1 - разметка будет прекращена.

6. Установка области для ошибок теста при включении питания

Две следующие функции применимы только в System/2, исключая модель 30, они дают возможность получать информацию о неработоспособных устройствах, установленную при выполнении теста включения питания (POST).

В большинстве моделей компьютеров возвращается (CF) = 1 как признак того, что функции не поддерживаются.

1. Получить область ошибок POST.

Параметры: (AX) = 2100H.
Результат: (ES:DI) - указатель на область ошибок,
(BX) - размер области (слов),
(AH) = 0 и (CF) = 0.

2. Записать код ошибки в область ошибок POST.

Параметры: (AX) = 2101H,
(BH) - код устройства,
(BL) - код ошибки.
Результат: (CF) = 0 и (AH) = 0 - нормальное выполнение,
(CF) = 1 и (AH) = 1 - переполнение таблицы ошибок.

7. Чтение и модификация системных параметров

Четыре подфункции данной функции поддерживаются только в PC Convertible. В других компьютерах возвращается (CF) = 1 и (AH) = 80H или 86H как признак того, что функция не поддерживается.

Параметры: (AH) = 40H,
(AL) - код подфункции:
0 - чтение системных параметров,
1 - модификация системных параметров,
2 - чтение параметров внутреннего модема,
3 - модификация параметров внутреннего модема,
(BX) - параметры для подфункций 1 и 3,
(CX) - параметры для подфункции 1.
Результат: (CF) = 0 - нормальное выполнение,
(AL) = 0 - нормальное выполнение,
(BX) - прочитанные параметры,
(CX) - прочитанные параметры для подфункции 0 или
(CF) = 1 и (AL) = 80H - флаг ошибки.

Системные параметры управляют перезапуском системы после выключения питания или сброса. Системные параметры имеют следующий формат (звездочкой отмечены значения, устанавливаемые после случайного выключения питания): (BX) бит 15 = 0 - режим "холодного" запуска, = 1 (*) - режим "горячего" запуска, бит 14 = 0 - запрет вывода сообщения о запуске, = 1 (*) - разрешение вывода сообщения, биты 13 - 12 - начальный режим дисплея: 00 - резервируется, 01 - монохромный 40*25 с использованием CGA (VGA или совместимого адаптера), 10 (*) - монохромный 80*25 с использованием CGA (VGA или совместимого адаптера), 11 - монохромный 80*25 с использованием монохромного адаптера, биты 11 - 10 - использование атрибутов VGA: 00 (*) - игнорировать атрибут интен-

сивности, 01 - использовать как подчеркивание, 10 - использовать для обратной контрастности, 11 - использовать для альтернативного набора знаков, бит 9 = 0 (*) - внутренний модем недоступен при автономном питании, 1 - внутренний модем доступен при автономном питании, бит 8 = 0 (*) - внешние линии связи недоступны при автономном питании, 1 - внешние линии связи доступны при автономном питании, биты 7 - 0 - резервируются. (CX) биты 15 - 8 - время (в минутах), в течение которого экран должен быть пустым при отсутствии активности клавиатуры, биты 7 - 0 - время (в минутах) до выключения компьютера при отсутствии активности клавиатуры.

Параметры внутреннего модема имеют следующий формат:

(BX) биты 15 - 14 - резервируются,
бит 13 = 0 (*) - ручной ответ,
1 - автоматический ответ,
биты 12 - 10 - контроль паритета и размер кадра:
000 - семибитовые данные с единичным битом,
001 - семибитовые данные с нулевым битом,
010 - семибитовые данные, четность,
011 (*) - семибитовые данные, нечетность,
100 - восьмибитовые данные,
иначе - резервируется,
биты 9 - 8 - скорость передачи:
00 - 110 бод,
01 - 300 бод,
10 (*) - 1200 бод,
11 - 2400 бод,
биты 7 - 0 - резерв.

8. Ожидание внешнего события

Все подфункции данной функции поддерживаются только в PC Convertible.
В других компьютерах возвращается (CF) = 1 и (AH) = 80H или 86H как признак того, что функция не поддерживается.

Внешнее событие может быть прерыванием или активностью канала НДП. До наступления ожидаемого события (состояния) выключается тактовый генератор.

Параметры: (AH) = 41H,
(AL) - код типа события:
00 - возврат при наступлении события,
01 - возврат при совпадении значений,
02 - возврат при несовпадении значений,
03 - возврат при ненулевом результате проверки битовой маски,
04 - возврат при нулевом результате проверки битовой маски,
ЮН - 14H - аналогично 00 - 04, но событие связывается
не с байтом памяти, а с портом ввода-вывода,

(BH) - значение для сравнения или маска для проверки,
(BL) - тайм-аут в единицах системного таймера (55 мсек),
(ES:DI) - указатель на проверяемый байт памяти (бит 4 AL равен 0),
(DX) - адрес порта ввода-вывода (бит 4 AL равен 1).
Результат: (CF) = 1 - тайм-аут.

Для кодов 01, 02, 11H и 12H байт, вызывающий событие, сравнивается с образцом инструкцией CMP, для кодов 03, 04, 13H и 14H - проверяется по маске инструкцией TEST.

9. Запрос сброса системы

Данная функция поддерживаются только в PC Convertible. В других компьютерах возвращается (CF) = 1 и (AH) = 80H или 86H как признак того, что функция не поддерживается.

Параметры: (AH) = 42H,
(AL) - код возобновления:
00 - при перезапуске используются системные параметры (см. функцию 40H),
01 - продолжение выполнения.
Результат: (CF) = 1 - функция не поддерживается или недоступна,
(AH) - модифицируется.

Управление немедленно возвращается лишь в том случае, если функция не поддерживается или недоступна; (CF)= 1 свидетельствует об этом. Иначе выключается питание с предварительной установкой режима, который будет использоваться при следующем включении компьютера. Если (AL) = 1, а параметры системны (см. функцию 40H) указывают на "горячий" запуск, то после включения питания будет продолжено выполнение программы, вызвавшей эту функцию.

Замечание: перед вызовом этой функции двигатели дискетных устройств должны быть выключены.

10. Чтение состояния системы

Данная функция поддерживаются только в PC Convertible.

В других компьютерах возвращается (CF) = 1 и (AH) = 80H или 86H как признак того, что функция не поддерживается.

Параметры: (AH) = 43H.
Результат: (CF) = 0 - нормальное выполнение,
(AH) - модифицируется,
(AL) - байт состояния системы
или (CF) = 1 - функция не поддерживается,

Функция возвращает байт состояния системы:

бит 7 = 1 - автономный источник питания,
6 = 1 - внешний источник питания,
5 = 1 - автономное питание потеряно(этом случае часы реального времени неработоспособны),
4 = 1 - включение питания вызвано сигналом от часов реального времени,
3 = 1 - внутренний модем включен,
2 = 1 - внешние линии связи включены,
1 - резервируется,
0 = 1 - адаптер дисплея (VGA) не подключен.

(См. функцию 8 прерывания 1 AH относительно сигнала от часов реального времени, включающего питание).

11. Включение или выключение питания внутреннего модема

Данная функция поддерживаются только в PC Convertible. В других компьютерах возвращается (CF) = 1 и (AH) = 80H или 86H как признак того, что функция не поддерживается.

Параметры: (AH) = 44H,
(AL) = 0 - выключить внутренний модем,
1 - включить внутренний модем.
Результат: (CF) = 1 - функция не поддерживается или запрос не выполнен,
(AL) = 80H - запрос не выполнен,
(CF) = 0 и (AL) = 0 - успешное выполнение.

При включении внутреннего модема автоматически устанавливается соответствующее состояние системы (см. функцию 40H).

12. Перехват клавиатурных кодов

Параметры: (AH) = 4FH,
(AL) - скан-код клавиши.
Результат: (CF) = 1 - скан-код заменен,
(AL) - новый скан-код
или (CF) = 0 - скан-код не изменился (AL сохраняется).

В некоторых компьютерах эта функция вызывается из обработчика клавиатурных прерываний (INT 9) после приема каждого знака от клавиатуры. Вызов функции позволяет заменить принятый скан-код до его обработки и размещения в буфере.

Замечания: это средство не поддерживается ранними версиями BIOS. В зависимости от версии BIOS (цаты создания) возвращается (CF) = 1 и (AH) = 80H или (CF) = 1 и (AH) = 86H. Функция 0C0H INT 15H позволяет определить, поддерживается ли данное средство.

BIOS, поддерживающие эту функцию, оставляют скан-коды неизменными.

13. Открыть устройство

Параметры: (AH) = 80H,
(BX) - идентификатор устройства,
(CX) - идентификатор процесса.
Результат: (CF) = 0 - нормальное выполнение,
1 - функция не поддерживается.

Функция предусмотрена для поддержки многозадачных операционных систем, позволяя открывать физические устройства для процессов. Обычно обработчик этого прерывания устанавливается операционной системой, которая ведет список активных процессов. Функция поддерживается (инициируется) не всеми версиями BIOS. Она не поддерживается в PC, PC-jr и в ранних версиях BIOS XT.

14. Закрыть устройство

Параметры: (AH) = 81H,
(BX) - идентификатор устройства,
(CX) - идентификатор процесса.
Результат: (CF) = 0 - нормальное выполнение,
1 - функция не поддерживается.

Функция предусмотрена для поддержки многозадачных операционных систем, позволяя отключать физические устройства от процессов. Обычно обработчик этого прерывания устанавливается операционной системой, которая ведет список активных процессов. Функция поддерживается (инициируется) не всеми версиями BIOS. Она не поддерживается в PC, PC-jr и в ранних версиях BIOS XT.

15. Прекратить программу устройства

Параметры: (AH) = 81H,
(BX) - идентификатор устройства.
Результат: (CF) = 0 - нормальное выполнение,
1 - функция не поддерживается.

Функция предусмотрена для поддержки многозадачных операционных систем, позволяя отключить физическое устройство от всех процессов. Обычно обработчик этого прерывания устанавливается операционной системой, которая ведет список активных процессов. Функция поддерживается (инициируется) не всеми версиями BIOS. Она не поддерживается в PC, PC-jr и в ранних версиях BIOS XT.

16. Ожидание события

Параметры: (AH) = 83H,
(AL) - код подфункции:
0 - установить промежуток времени,
1 - отменить установленный промежуток,
(ES:BX) - указатель на байт, используемый для установки события, состоящего в истечении указанного промежутка времени,
(CX:DX) - величина промежутка в мксек.
Результат: (CF) = 0 - нормальное выполнение,
1 - функция не поддерживается.

Функция поддерживается только в компьютерах, снабженных часами реального времени. В некоторых моделях (AT с BIOS от 1/10/84) поддерживается только подфункция 0.

После вызова функции управление немедленно возвращается программе пользователя, и, для того чтобы узнать об истечении установленного промежутка времени, она должна периодически просматривать байт ES:BX. Истечение промежутка времени отмечается установкой в 1 старшего бита указанного байта; программа пользователя должна сбросить этот бит перед вызовом подфункции 0. Часы реального времени обеспечивают кванты времени по 976 мксек. Старшая часть промежутка времени задается в (CX). Например, (CX) = 98H и (DX) = 9680H - определяют промежуток в 10 сек.

17. Поддержка координатных ручек

Параметры: (AH) = 84H,
(DX) - код подфункции:
0 - чтение текущей установки переключателей,
1 - чтение resistive inputs
Результат: (CF) = 1 - функция не поддерживается,
0 - нормальное выполнение,
для подфункции 0 биты 4 - 7 (AL) - установленные переключатели;
для подфункции 1 (AX) - значение A(x),
(BX) - значение A(y),
(CX) - значение B(x),
(DX) - значение B(y).

Функция предназначена для поддержки координатных ручек (Joystick) и реализована в BIOS не всех компьютеров. Она не поддерживается в PC, PC-jr и в ранних версиях BIOS XT.

18. Поддержка клавиши SysReq

Параметры: (AH) = 85H,
(AL) - код подфункции:
0 - запрос системы при нажатии клавиши,
1 - запрос системы при отпускании клавиши.
Результат: (CF) = 0 - нормальное выполнение,
1 - функция не поддерживается.

Функция, определенная в BIOS, не вызывает никаких действий, кроме очистки флага CF. Она позволяет операционной системе или пользователю определить обработчик клавиши SysReq. Этот обработчик будет вызываться из обработчика клавиатурного прерывания (INT 9) либо при нажатии клавиши SysReq, либо при ее отпускании.

Функция не поддерживается в PC, PC-jr и в ранних версиях BIOS XT.

19. Задержка

Параметры: (AH) = 86H,
(CX:DX) - время в мксек.
Результат: (CF) = 0 - нормальное выполнение,
1 - функция не поддерживается.

Функция поддерживается только в компьютерах, снабженных часами реального времени. Часы реального времени обеспечивают кванты времени по 976 мксек. Старшая часть промежутка времени задается в (CX). Например, (CX) = 98H и (DX) = 9680H определяют промежуток в 10 сек.

Управление возвращается после истечения указанного интервала времени. Функция не поддерживается в PC, PC-jr и в ранних версиях BIOS XT.

20. Перемещение блока памяти

Параметры: (AH) = 87H,
(CX) - размер перемещаемого блока памяти в словах
(максимум 8000H, что соответствует 64 Кб памяти),
(ES:SI) - указатель списка дескрипторов (см. ниже).
Результат: (CF) = 0 - нормальное выполнение
1 - ошибка,
(AH) - код завершения:
0 - успешное завершение,
1 - ошибка схем контроля,
2 - выполнение прекращено,
3 - неверный адрес памяти.

Функция поддерживается только BIOS компьютеров на основе микропроцессоров Intel 80286 и совместимых с ним (поддерживающих память 16М и привилегированный режим работы). Она позволяет переместить в область DOS блок памяти с абсолютным адресом, большим 1М.

ES:SI указывает на список из шести дескрипторов, каждый из которых имеет длину 8 байт. Формат дескриптора:

байты 0 - 1 - размер сегмента (слово),
байты 2 - 3 - младшее слово 24-битового адреса,
байт 4 - старший байт 24-битового адреса,
байт 5 - код права доступа,
байты 6 - 7 - резервируются.

Таблица дескрипторов, на которую указывает (ES:SI), содержит шесть следующих дескрипторов (в указанном порядке):

- пустой дескриптор, должен быть установлен пользователем на адрес 000000;
- дескриптор данной таблицы дескрипторов; устанавливается пользователем на адрес 000000; модифицируется BIOS;
- дескриптор перемещаемого блока памяти;
- дескриптор области для перемещения блока;
- дескриптор кодового сегмента для программы в привилегированном режиме; устанавливается пользователем на адрес 000000; модифицируется BIOS;
- дескриптор стека программы в привилегированном режиме; устанавливается пользователем на адрес 000000; модифицируется BIOS.

Пользователь устанавливает в значения, отличные от 0, только третий и четвертый дескрипторы таблицы. Они определяются следующим образом:

- поле границы сегмента должно содержать значение, не меньшее, чем 2**((CX)-1);
- поле адреса устанавливается в абсолютный адрес блока;
- поле права доступа должно быть установлено в 93Н, чтобы обеспечить доступ для чтения и для записи.

Функция переключает процессор в привилегированный режим и выполняет перемещение блока памяти. Во время перемещения запрещаются все прерывания, так что перемещение больших блоков может вызывать потерю сигналов прерывания.

21. Определение объема дополнительной памяти

Параметры: (АН) = 88Н.
Результат: (CF) = 0 - нормальное выполнение,
(АХ) - объем в килобайтах непрерывного участка ОЗУ, начинающегося с абсолютного адреса 100000Н
или (CF) = 1 - функция не поддерживается.

Функция поддерживается только BIOS компьютеров на основе микропроцессоров Intel 80286 и совместимых с ним (поддерживающих память 16М и привилегированный режим работы). Она возвращает объем памяти, распознаваемой BIOS при вклю-

чении питания. Эта память может использоваться только в привилегированном режиме работы процессора.

22. Переключение процессора в защищенный режим

Параметры: (AH) = 89H, (BH) - индекс в таблице описания прерываний, устанавливающий номера векторов прерывания от главного ПКП (соответствует IRQ 0); (BL) - индекс в таблице описания прерываний, устанавливающий номера векторов прерывания от подчиненного ПКП, (ES:SI) - указатель на таблицу дескрипторов (см. ниже). Результат: (AH) = 0 (и (CF) = 0) - успешное выполнение, (CF) = 0 и (AH) отлично от O - функция не выполнена; (CF) = 1 - функция не поддерживается.

Функция поддерживается только BIOS компьютеров на основе микропроцессоров Intel 80286 и совместимых с ним (поддерживающих память 16M и привилегированный режим работы). Нормальное выполнение функции переводит процессор в привилегированный режим и передает управление в сегмент, указанный пользователем. При этом разрушаются регистры AH, BP и все сегментные регистры.

(ES:SI) указывает на таблицу из восьми восьмибайтовых дескрипторов, формат которых приведен ниже.

Содержит следующие дескрипторы (в указанном порядке):

- пустой дескриптор должен быть установлен пользователем на адрес 000000;
- дескриптор данной таблицы дескрипторов; устанавливается пользователем на адрес 000000; модифицируется BIOS;
- дескриптор таблицы векторов прерываний; инициализируется пользователем;
- дескриптор сегмента данных; инициализируется пользователем;
- дескриптор дополнительного сегмента данных; инициализируется пользователем;
- дескриптор сегмента стека; инициализируется пользователем;
- дескриптор кодового сегмента; инициализируется пользователем;
- дескриптор временного кодового сегмента BIOS; устанавливается пользователем на адрес 000000; модифицируется BIOS.

Программе пользователя, которая получит управление, будут недоступны средства BIOS, поэтому она должна сама выполнять операции ввода-вывода. Векторы прерывания должны быть перемещены в резервную область памяти. Контроллеры прерываний должны быть переустановлены, чтобы они определяли векторы, не размещенные в резервной области. Обработчики прерываний должны быть инициализированы пользователем. Таблица описания прерываний не может быть перекрыта таблицей описания прерываний привилегированного режима BIOS.

23. Сообщение о занятости устройства

Параметры: (AH) = 90H,
(AL) - тип устройства (см. ниже).
Результат: (CF) = 0 - недостаточное время ожидания,
1 - достаточное время ожидания.

Функция 90Н вызывается для того, чтобы сообщить операционной системе, что логическое устройство ожидает обслуживания. Если процессу необходим доступ к некоторому устройству, а оно занято, то система может выполнять другие задачи, пока не освободится нужное устройство.

Тип устройства кодируется следующим образом:

- коды 00 - 7FH резервируются для последовательно используемых устройств (система может выдать запрос к такому устройству только в том случае, если предыдущий запрос к нему завершен);
- коды 80Н - OBFH резервируются для реентерабельных устройств (для такого устройства может одновременно существовать несколько незавершенных запросов на ввод-вывод; (ES:BX) используется для дифференциации запросов);
- коды 0C0H - OFFH резервируется для устройств и ситуаций, в которых не требуется завершения запроса после окончания времени ожидания (тайм-аут). В частности:

  00 - тайм-аут жесткого диска,
  01 - тайм-аут дискетного устройства,
  02 - клавиатура,
  03 - тайм-аут символьного устройства,
  80Н - сеть; (ES:BX) указывает на блок управления;
  OFDH - включение двигателя дискетного устройства,
  OFEH - устройство печати (тайм-аут).

Функция поддерживается не всеми компьютерами. Если она не поддерживается, то возвращается (CF) = 1 и (АН) = 80Н или 86Н в зависимости от модели.

24. Сообщение о готовности устройства

Параметры: (АН) = 91Н,
(AL) - тип устройства
Результат: (CF) = 0 - нормальное выполнение,
1 - ошибка.

Функция 91Н вызывается драйвером устройства, чтобы сообщить операционной системе, что устройство готово к обслуживанию (т.е. выполнило запрос). Например, обработчик клавиатурных прерываний (INT 9) может вызвать эту функцию, чтобы сообщить о том, что принят и обработан байт клавиатурных данных. Вызов функции устанавливает флаг завершения прерывания. Тип устройства - см. 16.21.

Функция поддерживается не всеми компьютерами. Если она не поддерживается, то возвращается (CF) = 1 и (АН) = 80Н или 86Н в зависимости от модели.

25. Получить параметры модели компьютера

Параметры: (АН) = ОСОН.
Результат: (CF) = 0 и (АН) = 0 - нормальное выполнение,
(ES:BX) - указатель на таблицу параметров
или (CF) = 1 - ошибка.

Возвращается указатель на таблицу:

байт 0 - длина таблицы в байтах (8);
байт 1 - код модели компьютера,
байт 2 - код исполнения (уточнение модели);
байт 3 - код версии BIOS;
байт 4 - код специальных средств;
байты 5 - 8 - резерв (нули).

Байт кода специальных средств имеет следующий формат:

бит 7 = 1 - BIOS жесткого диска использует канал 3 НДП;
бит 6 = 1 - установлен подчиненный ПКП;
бит 5 = 1 - установлены часы реального времени;
бит 4 = 1 - обработчик клавиатурных прерываний вызывает функцию (AH) = 4FH INT 15H;
бит 3 = 1 - поддерживается функция ожидания внешнего события;
биты 2 - 0 - резерв.

Функция поддерживается не всеми компьютерами. Если она не поддерживается, то возвращается (CF) = 1 и (AH) = 80H или 86H в зависимости от модели. Функция не поддерживается в PC, PC-jr и в ранних версиях BIOS XT.

Замечание: канал 3 КНДП может использоваться не только адаптером жесткого диска, но и другими адаптерами, в частности, он используется контроллером IBM локальной сети.

26. Получить сегмент расширения области данных BIOS

Следующая функция поддерживается только в System/2.

Параметры: (AH) = 0C1H.
Результат: (CF) = 0 - нормальное выполнение,
(ES) - сегментный адрес расширения области данных BIOS.

Если функция не поддерживается, то возвращается (CF) = 1 и (AH) = 80H или 86H в зависимости от модели.

16.27. Интерфейс с устройствами типа "мышь"

Восемь следующих функций предназначены для управления устройствами типа "мышь" и поддерживаются только в System/2. Если функция не поддерживается, то возвращается (CF) = 1 и (AH) = 80H или 86H в зависимости от модели.

1. Разрешение или запрет работы с устройством

Параметры: (AX) = 0C200H,
(BH) = 0 - запрет работы с устройством,
1 - разрешение работы.

Результат: (CF) = 0 - успешное выполнение,
= 1 - выход с ошибкой;
(AH) - состояние устройства:
0 - нет ошибок,
1 - неверная функция,
2 - неверные данные,
3 - ошибка интерфейса,
4 - Resend
5 - не установлен драйвер.

2. Сброс устройства

Параметры: (AX) = OC201H.
Результат: (CF) = 0 - успешное выполнение,
= 1 - выход с ошибкой;
(AH) - состояние устройства (как в 1) и
(BH) - идентификатор устройства,

Если операция завершается успешно, то устройство устанавливается в следующее состояние:

- работа запрещена,
- установлена частота 100 сообщений в секунду,
- установлено разрешение 4 точки на миллиметр,
- установлена линейная шкала 1:1,
- размер пакета данных остается тем же, что и перед вызовом этой функции.

Значение (BL) разрушается.

3. Установить частоту сообщений

Параметры: (AX) = OC202H,
(BH) - частота:
0 - 10 сообщений в секунду
1 - 20 сообщений в секунду
2 - 40 сообщений в секунду
3 - 60 сообщений в секунду
4 - 80 сообщений в секунду
5 - 100 сообщений в секунду
6 - 200 сообщений в секунду
Результат: (CF) = 0 - успешное выполнение,
= 1 - выход с ошибкой;
(AH) - состояние устройства (как в 1).

4. Установить разрешение

Параметры: (AX) = OC203H,
(BH) - разрешение:
0 - 1 точка на миллиметр
1 - 2 точки на миллиметр

2 - 4 точки на миллиметр
3 - 8 точек на миллиметр
Результат: (CF) = 0 - успешное выполнение,
= 1 - выход с ошибкой;
(AH) - состояние устройства (как в 1).

5. Получить ID устройства

Параметры: (AX) = OC204H.
Результат: (CF) = 0 - успешное выполнение,
. = 1 - выход с ошибкой;
(AH) - состояние устройства (как в 1),
(BH) - идентификатор устройства.

6. Инициализация интерфейса с устройством

Параметры: (AX) = OC205H,
(BH) - размер пакета данных:
O - резервируется,
иначе - число байтов в пакете (1 - 8).
Результат: (CF) = 0 - успешное выполнение,
= 1 - выход с ошибкой;
(AH) - состояние устройства (как в 1).

Если операция завершается успешно, то устройство устанавливается в следующее состояние:

- работа запрещена,
- установлена частота 100 сообщений в секунду,
- установлено разрешение 4 точки на миллиметр,
- установлена линейная шкала 1:1.

7. Расширения команд инициализации

Параметры: (AX) = OC206H,
(BH) = 0 - получить расширенное состояние,
1 - установить масштаб 1:1,
2 - установить масштаб 2:1.
Результат: (CF) = 0 - успешное выполнение,
= 1 - выход с ошибкой;
(AH) - состояние устройства (как в 1).

Расширенное состояние для подфункции (BH) = 0 возвращается в регистрах (BL), (CL) и (DL):

(BL) бит 7 - резерв (0);
6 = 0 - режим потока (stream),
1 - удаленный режим (remote);

5 = 0 - разрешение,
1 - запрет;
4 = 0 - масштаб 1:1,
1 - масштаб2:1;
3 - резерв (0);
2 = 1 - нажаталевая кнопка,
1 - резерв (0);
0 = 1 - нажатаправаякнопка,
(CL) =00 - 1 точка на миллиметр,
01 - 2точки на миллиметр,
02 - 4точки на миллиметр,
03 - 8 точек на миллиметр;
(DL) - число сообщений в секунду (10, 20, 40, 60, 80, 100 или 200).

8. Инициализация драйвера

Параметры: (AX) = OC207H
(ES:BX) - адрес точки входа в драйвер
Результат: (CF) = 0 - успешное выполнение,
= 1 - выход с ошибкой;
(AH) - состояние устройства (как в 1).

Пользователь должен определить модуль, который получает управление всякий раз, когда становятся доступными какие-либо данные устройства. Данная функция устанавливает адрес драйвера для BIOS.

Драйвер вызывается инструкцией межсегментного вызова и получает параметры в стеке:

слово 1 - состояние:
бит 7 = 1 - переполнение по координате Y,
бит 6 = 1 - переполнение по координате X,
бит 5 = 1 - отрицательные данные по координате Y,
бит 4 = 1 - отрицательные данные по координате X,
бит 3 - резерв (всегда 1),
бит 2 - резерв (всегда 0),
бит 1 = 1 - нажата правая кнопка,
бит 0 = 1 - нажата левая кнопка,
биты 15 - 8 - нули;
слово 2 - байт данных по координате X (биты 15 - 8 - нули);
слово 3 - байт данных по координате Y (биты 15 - 8 - нули);
слово 4 - нули.

Выход из драйвера должен быть оформлен как межсегментный возврат, информация в стеке не должна изменяться.

28. Разрешение или запрещение "сторожевого пса"

Следующая функция поддерживается только в System/2, исключая модель 30. Если функция не поддерживается, то возвращается (CF) = 1 и (AH) = 80H или 86H в зависимости от модели.

Параметры: (AH) = OC3H,
(AL) = 0 - запрет тайм-аута "сторожевого пса",
1 - разрешение тайм-аута,
(BX) - счетчик (1 - 255).
Результат: (CF) = 0 - нормальное выполнение,
1 - выход с ошибкой.

"Строжевой пес" - это таймер, сигнализирующий о необработанных прерываниях.

29. Выбор программируемых режимов

Следующие три функции поддерживаются только в System/2.
Если функции не поддерживаются, то возвращается (CF) = 1 и (AH) = 80H или 86H в зависимости от модели.

1. Получить базовый адрес регистров адаптера POS.

Параметры: (AX) = OC400H.
Результат: (CF) = O, (AL) = O,
(DX) - базовый адрес регистров POS.

2. Разрешить установку платы.

Параметры: (AX) = OC401H,
(BL) - номер платы.
Результат: (CF) = 0 - нормальное выполнение,
1 - выход с ошибкой.

3. Разрешить работу адаптера.

Параметры: (AX) = OC402H.
Результат: (CF) = 0 - нормальное выполнение,
1 - выход с ошибкой.

Обслуживание клавиатуры (прерывание 16H).

Функции BIOS для работы с клавиатурой перечислены на Рис. 3. Только функции 0 - 2 поддерживаются любой версией BIOS. Остальные функции требуют специального оборудования или расширения интерфейса с аппаратурой. В частности, функции

1ОН, 11Н и 12Н предназначены для поддержки клавиатур с расширенными функциональными возможностями, позволяющими получать последовательность скан-кодов при нажатии или отпускании одной клавиши. Для таких клавиатур скан-коды подразделяются на три категории:

1. Если некоторый знак порождается нажатием (или отпусканием) только одной клавиши, то этот скан-код такой клавиши обрабатывается обычным образом.
2. Если несколько клавиш генерируют один и тот же знак, то только одна из них генерирует стандартный скан-код, соответствующий этому знаку. Другие клавиши генерируют уникальные последовательности скан-кодов, в каждую из которых входит стандартный скан-код знака. Поэтому система может различать нажатые клавиши.
3. Новым клавишам присвоены уникальные скан-коды.

Во всех моделях компьютеров функции 0 и 1 возвращают только стандартные коды знаков и скан-коды клавиш, удаляя из буфера дополнительную информацию. Этим достигается независимость программ от установленной клавиатуры. При использование расширенной клавиатуры не следует считать, что в программу попадает вся информация из буфера. Точную копию буфера можно получить с помощью функций ЮН и 11Н.

Чтобы определить, что функции ЮН - 12Н поддерживаются BIOS, можно, воспользовавшись функцией 5, занести в буфер код OFFH со скан-кодом OFFH. Если функция 5 вернет (AL) = 0, то код успешно записан в буфер, иначе при отсутствии переполнения буфера функции 5, ЮН, 11Н и 12Н не поддерживаются. Затем нужно вызывать функцию ЮН, и если ни одна из 16 попыток ее вызова не вернет (АХ) = OFFFFH, то функции ЮН -12Н не поддерживаются.

код	функция
00	чтение знака
01	проверка наличия знака
02	состояние регистров клавиатуры
03	установка частоты повторений
04	управление звуковым сигналом
05	запись в буфер клавиатуры
10	расширенная функция чтения
11	расширенная проверка наличия знака
12	расширенное состояние регистров

1. Чтение знака

Параметры: (АН) = 0.
Результат: (AL) - код знака,
(АН) - скан-код клавиши.

Код знака и скан-код клавиши удаляются из буфера BIOS возвращаются в (AX). Если буфер BIOS пуст, функция ждет ввода знака с клавиатуры. В некоторых моделях до перехода к ожиданию вызывается функция 90H прерывания 15H, чтобы сообщить (многозадачной) операционной системе, что она может переключиться на другую задачу до получения знака. В этих же моделях после получения кода от клавиатуры система информируется об этом вызовом функции 91H INT 15H.

Замечание: знаки расширенного кода (в частности, управляющие символы) возвращаются в виде скан-кода в (AH) с (AL) = 0.

2. Проверка наличия знака

Параметры: (AH) = 1.
Результат: (ZF) = 1 - буфер клавиатуры пуст,
(ZF) = 0 - в буфере есть знаки и
(AL) - код первого знака из буфера,
(AH) – scan-код клавиши.

В отличие от функции 0 информация, возвращаемая в (AX), не удаляется из буфера. Если в буфере BIOS нет знаков, немедленно возвращается (ZF) = 1.

3. Состояние регистров клавиатуры

Параметры: (AH) = 2.
Результат: (AL) - первый байт состояния регистров клавиатуры,
(AH) разрушается.

Функция возвращает в (AL) в точности ту же информацию, которая содержится в байте памяти с абсолютным адресом 0417H.

4. Установка частоты повторений.

Данная функция требует специального адаптера клавиатуры и поддерживается только в некоторым моделях компьютеров. Подфункции (AL) = 0,...,4 поддерживаются только в PC-jr, подфункция (AL) = 5 - в AT и в System/2.

Параметры: (AH) = 3,
(AL) - код подфункции:
0 - установить умолчание для состояния регистров клавиатуры,
1 - увеличить начальную задержку,
2 - уменьшить частоту повторов вдвое,
3 - увеличить начальную задержку и уменьшить вдвое частоту повторов,
4 - отключить повторы,
5 - установить начальную задержку и частоту повторов;

(BL) - частота повторов (для подфункции 5),
(BH) - начальная задержка (для подфункции 5).

Подфункция 1 устанавливает регистры клавиатуры в то состояние, в какое они устанавливаются сбросом. Остальные подфункции управляют временем между повторными выдачами скан-кода нажатой клавиши и задержкой до первого повторения.

Подфункция 5 допускает значения (BL) от 0 до 31, что соответствует интервалу между повторами от 30 мсек до 2 мсек (большее значение (BL) соответствует меньшему промежутку между повторами). (BH) указывает промежуток времени до первого повторения. Допускаются значения от 0 до 3, и соответствующие промежутки от 0.25 сек до 1 сек.

5. Управление звуковым сигналом

Данная функция требует специального адаптера клавиатуры и поддерживается только в некоторых моделях компьютеров (PC-jr, PC Convertible).

Параметры: (AH) = 4,
(AL) = 0 - отключить звуковой сигнал,
1 - включить звуковой сигнал.

Звуковой сигнал выдается при переполнении внутреннего буфера клавиатуры.

6. Запись в буфер клавиатуры

Данная функция позволяет программе заносить знаки в буфер BIOS так, как если бы эти знаки были получены из клавиатуры. Функция не поддерживается в PC, PC-jr, PC Convertible и ранними версиями BIOS XT и AT.

Параметры: (AH) = 5,
(CH) - скан-код,
(CL) - код знака.
Результат: (AL) = 0 - успешное выполнение,
1 - переполнение буфера.

7. Расширенная функция чтения

Параметры: (AH) = 10H.
Результат: (AL) - код знака,
(AL) - скан-код клавиши.

Эта функция, не реализованная во многих компьютерах, выполняется аналогично функции 0, но возвращает точную копию информации в буфере BIOS (см. функцию 1), Функция не поддерживается в PC, PC-jr, PC Convertible и ранними версиями BIOS XT и AT.

8. Расширенная проверка наличия знака

Параметры: (AH) = 11H.
Результат: (ZF) = 1 - буфер клавиатуры пуст,
(ZF) = 0 - в буфере есть знаки и
(AL) - код первого знака из буфера,
(AH) - scan-код клавиши.

Эта функция, не реализованная во многих компьютерах, выполняется аналогично функции 1, но возвращает точную копию информации в буфере BIOS (см. функцию 1). Функция не поддерживается в PC, PC-jr, PC Convertible и ранними версиями BIOS XT и AT.

9. Расширенное состояние регистров

Параметры: (AH) = 12H.
Результат: (AL) - первый байт состояния регистров клавиатуры,
(AH) - дополнительный байт состояния регистров клавиатуры.

Функция возвращает в (AL) в точности ту же информацию, которая содержится в байте памяти с абсолютным адресом 0417H, а в (AH) - ту же информацию, которая содержится в байте памяти с абсолютным адресом 0418H. Функция не поддерживается в PC, PC-jr, PC Convertible и ранними версиями BIOS XT и AT.

Вывод на печатающее устройство (прерывание 17H).

BIOS поддерживает три следующие функции устройств печати:
(AH) = 0 - вывод знака,
(AH) = 1 - инициализация порта устройства печати,
(AH) = 2 - чтение состояния.

1. Вывод знака

Параметры: (AH) = 0,
(AL) - знак для вывода,
(DL) - номер устройства печати.
Результат: (AH) - состояние окончания функции (см. 18.3).

Значение (DL) используется как индекс при выборе базового адреса портов устройства печати из таблицы с абсолютным адресом 0408H. Значение (DL) должно быть меньше числа подключенных устройств печати и во всех случаях не может быть больше трех.

2. Инициализация порта устройства печати.

Параметры: (AH) = 1,
(DL) - номер устройства печати.
Результат: (AH) - состояние окончания функции (см. 18.3).

Значение (DL) используется как индекс при выборе базового адреса портов устройства печати из таблицы с абсолютным адресом 0408Н. Значение (DL) должно быть меньше числа подключенных устройств печати и во всех случаях не может быть больше трех.

3. Получить состояние

Параметры: (АН) = 2,
(DL) - номер устройства печати.
Результат: (АН) - состояние устройства:
бит 7 = 1 - устройство неготово,
бит 6 = 1 - подтверждение,
бит 5 = 1 - нет бумаги,
бит 4 = 1 - выбор устройства,
бит 3 = 1 - ошибка вывода,
бит 0 = 1 - тайм-аут,
биты 2 и 1 резервируются.

Значение (DL) используется как индекс при выборе базового адреса портов устройства печати из таблицы с абсолютным адресом 0408Н. Значение (DL) должно быть меньше числа подключенных устройств печати и во всех случаях не может быть больше трех.

Обслуживание часов реального времени (прерывание 1АН).

В таблице перечислены функции BIOS, связанные с таймером и с часами реального времени. Только функции 0 и 1 доступны во всех моделях компьютеров, остальные требуют установки часов реального времени.

код	функция
00	чтение счетчика циклов таймера
01	установка счетчика циклов таймера
02	получить реальное время
03	установить реальное время
04	получить реальную дату
05	установить реальную дату
06	установить сигнал тревоги
07	сбросить установку сигнала тревоги
08	установить время включения питания
09	получить время и состояние сигнала тревоги
ОА	чтение счетчика дней
0В	установка счетчика дней
80	установить звуковой генератор

Функции даты и времени.

1. Чтение счетчика циклов таймера

Параметры: (AH) = 0.
Результат: (CX) - младшее слово счетчика,
(DX) - старшее слово счетчика,
(AL) - флаг перехода через сутки.

Обработчик прерывания от системного таймера (INT 8) подсчитывает количество случившихся прерываний в двойном слове памяти с адресом 0470H. Данная функция возвращает накопленное значение и сбрасывает его в 0. В регистре AL возвращается О, если содержимое счетчика не превысило значения, соответствующего 24 часам (при достижении этого значения счетчик сбрасывается), иначе возвращается (AL) = 1.

Поскольку содержимое счетчика сбрасывается при каждом обращении, то двукратная установка флага перехода через сутки практически невероятна, поскольку, например, каждое обращение к дискетному устройству содержит вызов данной функции.

2. Установка счетчика циклов таймера

Параметры: (AH) = 1,
(CX) - младшее слово счетчика,
(DX) - старшее слово счетчика.
Результатов нет.

Выполнение функции сбрасывает флаг перехода через сутки. Отметим, что максимальное значение счетчика циклов таймера равно 1800BOH.

Установка счетчика циклов таймера изменяет системное время, но не дату.

3. Получить реальное время

Параметры: (AH) = 2.
Результат: (CF) = 1 - часы не установлены или выключены,
(CF) = 0 - нормальное выполнение,
(CH) - часы,
(CL) - минуты,
(DH) - секунды,
(DL) = 0 - установлен режим 24-часовых суток,
1 - установлен режим 12-часовых суток.

Функция поддерживается не всеми компьютерами (AT, System/2), и даже для них не всегда поддерживается режим 12-часовых суток.

Результат возвращается в упакованном десятичном формате.

Например, (CX) = 0904H, (DH) = 12H обозначает 9 часов 4 минуты 12 секунд.

Часы реального времени имеют автономное питание и работают даже при выключении компьютера. Они используются при включении питания для установки системной даты и системного времени, но в дальнейшем дата и время определяются на основании счетчика таймера. Поэтому результат этой функции необязательно соответствует результату функции 0.

4. Установить реальное время

Параметры: (AH) = 3,
(CH) - часы,
(CL) - минуты,
(DH) - секунды,
(DL) = 0 - установить режим 24-часовых суток,
1 - установить режим 12-часовых суток.
Результат: (CF) = 0 - нормальное выполнение,
(CF) = 1 - часы не установлены или выключены.

Функция поддерживается не всеми компьютерами (AT, System/2), и даже для них не всегда поддерживается режим 12-часовых суток.

Параметры задаются в упакованном десятичном формате. Например, (CX) = 0904H, (DH) = 12H обозначает 9 часов 4 минуты 12 секунд.

5. Получить реальную дату

Параметры: (AH) = 4.
Результат: (CF) = 1 - часы не установлены или выключены,
(CF) = 0 - нормальное выполнение,
(CH) - век (XIX или XX),
(CL) - год,
(DH) - месяц,
(DL) - день.

Функция поддерживается не всеми компьютерами (AT, System/2).

Результат возвращается в упакованном десятичном формате. Например, (CX) = 1988H, (DX) = 1127H обозначает 27.11.1988. Результат данной функции не обязательно соответствует результату функции 1 (см. 20.3).

6. Установить реальную дату

Параметры: (AH) = 5,
(CH) - век (XIX или XX),
(CL) - год,
(DH) - месяц,
(DL) - день.

Результат: (CF) = 0 - нормальное выполнение,
(CF) = 1 - часы не установлены или выключены.

Функция поддерживается не всеми компьютерами (AT, System/2).

Параметры задаются в упакованном десятичном формате. Например, (CX) = 1988H, (DX) = 1127H обозначает 27.11.1988.

7. Установить сигнал тревоги

Параметры: (AH) = 6,
(CH) - часы,
(CL) - минуты,
(DH) - секунды.
Результат: (CF) = 0 - нормальное выполнение,
(CF) = 1 - часы не установлены или выключены.

Функция поддерживается не всеми компьютерами (AT, System/2).

Параметры задаются в упакованном десятичном формате. Например, (CX) = 0904H, (DH) = 12H обозначает 9 часов 4 минуты 12 секунд.

Установленный сигнал тревоги вызывает программное прерывание 4AH, обработчик которого может быть установлен пользователем. BIOS не сбрасывает сигнал тревоги при выполнении прерывания.

8. Сбросить сигнал тревоги.

Параметры: (AH) = 7.
Результат: (CF) = 0 - нормальное выполнение,
(CF) = 1 - часы не установлены или выключены.

Функция поддерживается не всеми компьютерами (AT, System/2).

Если сигнал тревоги не был установлен, то не выполняются никакие действия.

9. Установить время включения питания

Параметры: (AH) = 8,
(CH) - часы,
(CL) - минуты,
(DH) - секунды.
Результат: (CF) = 0 - нормальное выполнение,
(CF) = 1 - часы не установлены, или выключены, или уже установлен сигнал тревоги.

Функция требует установки не только часов реального времени, но и другого специального оборудования и поддерживается только в компьютерах, совместимых с System/2.

10. Получить время и состояние сигнала тревоги

Параметры: (AH) = 9.
Результат: (CF) = 1 - часы не установлены или выключены,
(CF) = 0 - нормальное выполнение,
(CH) - часы,
(CL) - минуты,
(DH) - секунды,
(DL) - состояние сигнала тревоги:
0 - не установлен,
1 - установлен, но не включает питание,
2 - установлен и включает питание.

Функция требует установки не только часов реального времени, но и другого специального оборудования и поддерживается только в компьютерах, совместимых с System/2.

Время возвращается в упакованном десятичном формате (см. 20.3).

11. Чтение счетчика дней

Параметры: (AH) = OAH.
Результат: (CF) = 1 - функция не поддерживается,
(CF) = 0 - нормальное выполнение,
(CX) - счетчик дней от 1.01.1980.

Функция не требует установки часов реального времени, но поддерживается не всеми компьютерами и версиями BIOS. Счетчик дней устанавливается в 0 при включении питания.

12. Установка счетчика дней

Параметры: (AH) = OBH,
(CX) - счетчик дней от 1.01.1980.
Результат: (CF) = 0 - нормальное выполнение,
1 - функция не поддерживается.

Функция не требует установки часов реального времени, но поддерживается не всеми компьютерами и версиями BIOS. Счетчик дней устанавливается в 0 при включении питания.

13. Установить звуковой генератор.

Функция применяется только в PC-jr.

Параметры: (AH) = 80H,
(AL) - источник звука:
0 - канал 2 таймера 8253,
1 - входной сигнал от кассетного устройства,
2 - линия "Audio In" в канале ввода-вывода,
3 - микросхема тонального генератора.
Результат: (CF) = 0.

В других компьютерах возвращается (CF) = 1 как признак того, что функция не поддерживается.

Приложение 9. Работа с портами ввода-вывода.

Несколько замечаний по работе с портами.

В микропроцессорах Intel изначально определены две команды для работы с портами: IN (для вывода из порта) и OUT (для ввода в порт). Начиная с 286-го процессора, появились также команды строковой пересылки INS и OUTS (см. главу 26), их мы здесь не рассматриваем. Есть две разновидности команд IN и OUT. Если номер порта не превышает FFH, то используется прямая адресация: IN AL,port, IN AX,port, OUT port,AL, OUT port,AX. Если номер порта превышает FFH, то используется косвенная адресация через регистр DX: IN AL,DX, IN AX,DX, OUT DX,AL, OUT DX,AX.

После выполнения каждой команды IN/OUT, вообще говоря, требуется некоторая задержка, т.к. внешнее устройство может не успеть среагировать, а последующая команда выполнится не вовремя. Известно также, что такой задержки не требует видеосистема. Рекомендуется делать задержку вида:

```
            JMP   MET
MET:
```

или просто JMP SHORT $+2. В некоторых ситуациях реакция оборудования проверяется по содержимому, какого-либо порта. При программировании некоторых устройств следует учесть и задержки, связанные с работой этих устройств. Это, в частности, касается и работы с гибким диском. Довольно часто в задержке нет необходимости, но познается это на практике.

Работа с клавиатурой.

Управление клавиатурой в машинах класса AT построено на основе микроконтроллера Intel 8042. Контроллер клавиатуры выполняет следующие действия:

- прием данных от клавиатуры;
- проверка четности поступившего байта;
- трансляция (кодирование) полученной информации;
- помещение байта данных в выходной буфер и извещение процессора;
- передача байта во входной буфер клавиатуры.

Для каждой клавиши клавиатурой вырабатывается два кода: код нажатия и код прерывания (клавишу отпустили). Коду прерывания предшествует код F0h. Контроллер преобразует коды от клавиатуры в scan-коды. Код F0h (при отпускании) преобразуется в 7-й бит scan-кода. Некоторые клавиши, однако, могут быть запрограммированы только на посылку клавиши нажатия. Кроме того, при длительном удержании клавиши начинают посылать повторные коды нажатия.

Регистр состояния контроллера клавиатуры. Доступен по чтению по адресу 64Н. Биты регистра:

0 - равен 0, если выходной буфер клавиатуры пуст.

1 - состояние входного контроллера клавиатуры. Равен нулю, если буфер пуст.

2 - отражает тот факт, что тестирование клавиатуры прошло.

3 - указывает контроллеру, как интерпретировать полученный байт. Если бит 1, то байт считается данным, если 0 - байт считается командой.

4 - если бит, равен 0, то клавиатура заблокирована.

5 - устанавливается в 1, если передача байта в клавиатуру не была завершена.

6 - устанавливается в 1, если прием байта от клавиатуры не был закончен в заданный промежуток времени.

7 - если значение равно 1, то последний принятый байт содержал четное число единиц, т.е. произошла ошибка.

Входной буфер контроллера. Доступен по записи: 60h - запись данных, 64Н - запись команды.

Выходной буфер контроллера. Доступен по чтению по адресу 60Н.

Входной порт контроллера клавиатуры. Доступен по чтению по команде СОН. Биты:

0-3 - резерв.

4 - указывает количество RAM на системной плате.

5 - значение переключателя фирмы-изготовителя.

6 - определяет тип видеоадаптера.

7 - указывает, заблокирована или нет клавиатура.

Данный порт устарел и данные, возможно, недостоверны.

Выходной порт контроллера клавиатуры. Доступен по чтению (команда DOH) и по записи (команда D1H). Биты:

0 - подсоединен к линии сброса системы, изменение приводит к сбросу.

1 - определяет состояние дополнительной адресной линии А20.

2-3 - значения не определены.

4 - содержит 1, если выходной буфер полон.

5 - равен 1, если входной буфер пуст.

6 - определяет значение линии часов при передаче данных в клавиатуру.

7 - определяет значение линии данных при передаче их в клавиатуру.

Порт состояния входных линий. Значение может быть считано по команде Е0Н.

0 - состояние входной линии часов.

1 - состояние входной линии данных.

2-7 не определены.

Команды контроллера клавиатуры.

Как уже упоминалось, команды передаются через порт 64Н. Если у команды существует параметр, то он помещается сразу за посылкой команды в порт 60Н.

Команда 20Н. После этой команды контроллер записывает значение управляющего байта в выходной буфер.

Команда **60H.** Запись управляющего байта контроллера (поместить в порт 60H). Значение битов:

0 - значение 1 приводит к генерированию прерывания от контроллера клавиатуры, если выходной буфер содержит данные.

1 - обычно 0.

2 - заменяет собой соответствующий бит управляющего регистра контроллера.

3 - при установке в **1,** переключатель замка клавиатуры перестает действовать.

4 - при установке в 1 клавиатура отключается.

5 - режим интерфейса. При установке в I не выполняется проверка четности, и не преобразуются scan-коды.

6 - бит совместимости.

7 - 0.

Команда AAH. Внутренний тест контроллера. Если ошибок нет, то в выходной буфер посылается 55h. После данной команды следует восстановить управляющий **байт** (см. **команды 20h и** 60h).

Команда ABH. По команде выполняется тест интерфейса между контроллером и клавиатурой. Результат помещается в выходной буфер:

00 - ошибок нет,

01 - сигнал линии часов завис в нижнем положении,

02 - сигнал линии часов завис в верхнем положении,

03 - сигнал линии данных завис в нижнем положении,

04 - сигнал линии данных завис в верхнем положении.

Команда ACH. Выполняется диагностический дамп **16-байтной** памяти контроллера, текущего состояния входного и выходного порта, а также слова состояния контроллера. Дамп выдается в виде последовательности scan-кодов. В конце выдается символьная строка, идентифицирующая фирму-изготовитель.

Команда **ADH.** Блокировка клавиатуры.

Команда AEH. Разблокировка клавиатуры.

Команда COH. Чтение входного порта. Значение порта помещается в выходной буфер.

Команда **D0H.** Чтение выходного порта. Значение порта помещается в выходной буфер.

Команда D1H. Помещает **байт** в выходной порт контроллера клавиатуры.

Команда **E0H.** Команда предназначена для помещения информации о состоянии входных линий **T0 и** T1 из порта состояния входных линий в выходной буфер.

Команды **F0H-FFH** - сброс выходного порта.

Команды управления клавиатурой.

Команды передаются через входной буфер **60H.** Если необходим параметр, то он передается после подтверждения получения команды. Подтверждением того, что клавиатура приняла команду, является чтение процессором из порта **60h** байта ответа клавиатуры. Ответ посылается **на** все команды, кроме **EEH** и FEH.

Команда EDH. Установка и сброс индикатора состояния. Следующий за командой параметр представляет собой байт, первые 3 бита которого определяют состояние индикаторов, а остальные равны **0: 0 - Scrol** Lock, **1 - Num** Lock, 2 - Caps Lock.

Команда EEH. На данную команду клавиатура откликается такой же командой (эхо). Используется для диагностики.

Команды EFH, F1H. Недействительные команды.

Команда FOH. Установка или запрос таблицы скан-кодов. Входной параметр: 0 - получить номер таблицы, 1 - установить таблицу 1, 2 - установить таблицу 2, 3 - установить таблицу 3.

Команда F2H. Прочитать идентификатор. Вначале посылается подтверждение, а затем идентификатор - 83ABH.

Команда F3H. Задание частоты повтора кода клавиш и начальной задержки. Используются биты 0-6:

частота = (1+2*b6+b5)*250;

повтор = 4.17*(8+4*b2+2*b1+b0)*2*(2*b4+b3).

Команда F4H. Включение клавиатуры. После этой команды клавиатура посылает подтверждение, чистит свой внутренний буфер и переходит в состояние ожидания.

Команда F5. По этой команде все характеристики устанавливаются в исходные, затем клавиатура блокируется и ждет дальнейших команд.

Команда F6h. Аналогична предыдущей, но клавиатура не блокируется.

Команды F7h-FAh. Характер реакции клавиш:

F7h - все клавиши повторяемые,

F8h - все клавиши посылают код нажатия и код прерывания,

F9h - все клавиши посылают только код нажатия,

FAh - все клавиши повторяемые и посылают коды нажатия и прерывания.

Команды FBh-FDh. Определение действия отдельных клавиш.

FBh - клавиша повторяемая,

FCh - клавиша посылает код нажатия и прерывания,

FDh - клавиша посылает только код нажатия.

При посылке такой команды следует послать scan-код клавиши (таблица 3).

Команда FEh. Повторить передачу - клавиатура повторяет передачу.

Команда FFh. Сброс клавиатуры. Выполняется внутренний тест и посылает AAh если все удачно, FCh - ошибка.

В качестве примера работы с клавиатурой ниже мы приводим программу (Рис. P12.1), которая несколько раз периодически зажигает и тушит индикаторы клавиатуры. Еще один пример использования изложенного выше материала Вы можете найти в главе 23, где приводится простая программа тестирования клавиатуры.

```
CODE  SEGMENT
      ASSUME CS:CODE, DS:DATA, SS:STAK
BEGIN:
      MOV   AX, DATA
      MOV   DS,AX
      MOV   _CL, 0
      MOV   COUNT,0
      MOV   _AL, 0
```

```
;отключить прерывания клавиатуры на уровне контроллера
;прерываний, другие прерывания работают исправно
      MOV   AL,00000010B
      OUT   21H,AL
LOO:
;ждать, когда освободится входной буфер
      CALL WAIT_IN_BUF
;послать команду
      MOV AL,0EDH
      OUT 60H,AL
;ждать подтверждение
      CALL WAIT_OUT
      IN   AL,60H
;правильное ли подтверждение?
      CMP AL,0FAH
      JNZ   ERR1
;теперь ждем, когда можно посылать параметр
      CALL WAIT_IN_BUF
      MOV   AL,00000001B
      MOV   CL,_CL
;переход к следующему индикатору
      SHL   AL,CL
      OR    _AL,AL
      MOV   AL,_AL
      OUT   60H,AL
;небольшая задержка
      CALL DELAY
      INC   _CL
;все ли индикаторы зажгли?
      CMP  _CL, 3
      JNZ   LOO
      MOV   _CL,0
      MOV  _AL, 0
      CALL WAIT_IN_BUF
;теперь погасим все индикаторы
      MOV AL,0EDH
      OUT 60H,AL
      CALL WAIT_OUT
      IN   AL,60H
      CMP AL,0FAH
      JNZ   ERR1
      CALL WAIT_IN_BUF
      MOV   AL,0
      OUT   60H,AL
      INC   COUNT
```

```
;не пора ли заканчивать?
        CMP   COUNT,20
        JZ    _END
        CALL  DELAY
        JMP   LOO
ERR:
        POP   AX
ERR1:
        MOV   AL,0
        OUT   21H,AL
        LEA   DX,MES
        INT   21H
        MOV   AX,4C01H
        INT   21H
_END:
        MOV   AL,0
        OUT   21H,AL
        MOV   AX,4C00H
        INT   21H
;ждать, когда освободится входной буфер
WAIT_IN_BUF PROC
        XOR CX,CX
T1:
        IN AL,64H
        TEST AL,2
        LOOPNZ  T1
        JNZ  ERR
        RETN
WAIT_IN_BUF ENDP
;ждать, когда заполнится выходной буфер
WAIT_OUT PROC
T3:
        IN AL,64H
        TEST AL,1
        JZ   T3
        RETN
WAIT_OUT ENDP
DELAY  PROC
        XOR CX,CX
LL1:
        PUSH  CX
        MOV   CX,50
LL2:
        LOOP  LL2
        POP   CX
```

```
      LOOP LL1
      RETN
DELAY ENDP
CODE ENDS
DATA SEGMENT
_CL   DB  ?
_AL   DB  ?
COUNT DW  ?
MES   DB   'Произошла ошибка клавиатуры!',13,10,'$'
DATA ENDS
STAK SEGMENT STACK
      DB 50 DUP(?)
STAK ENDS
      END BEGIN
```

Рис. P12.1. Пример простой программы программирования клавиатуры.

Порты для работы с параллельным интерфейсом.

В главе 7 мы довольно полно осветили вопрос работы с параллельным портом (с принтером). Здесь мы ограничимся кратким справочным материалом.

BIOS поддерживает до 3-х параллельных портов. При обнаружении порта адрес его регистра данных записывается, начиная с адреса **0:408h,** и ему присваивается имя **LPTn, где n= 1,2,3.** Ниже представлена адресация регистров параллельных портов.

Порт	Регистр данных	Регистр состояния	Регистр управления
LPT1	3BCH	3BDH	3BEH
LPT2	378H	379H	37AH
LPT3	278H	279H	27AH

Описание регистров.

Регистр данных.
Биты **0-7** - значение передаваемого или считанного байта.
Регистр состояния.
Биты:
0-1 **- резерв,**
2 - 0 если устройство подтвердило прием предыдущего байта.
3 - 0 при выработке принтером сигнала ошибки.
4 - 1 если данное устройство выбрано.
5 **- 1** если получен сигнал конец бумаги.
6 - 0 - устройство готово к приему.
7 - 0 - устройство занято.

Регистр управления.

Биты:

0 - сигнал строба, когда принимает значение 1, данные могут считываться с линий данных.

1 - если 1, то после печати каждой строки принтер автоматически будет переходить на следующую строку.

2 - 0 - инициализация принтера.

3 - 1 - устройство считается выбранным.

4 - 1 - параллельный порт посылает сигнал прерывания.

5 - 0 - операция записи, 1 - операция считывания.

Если на Вашем компьютере к параллельному порту подключен принтер, то Вам вполне достаточно прерывания 17H для полного управления принтером. Описание регистров, представленное нами, может Вам понадобиться, если Вы решили использовать параллельный порт для других целей (например, обмен информацией между компьютерами). Программа работы с принтером - см. главу 7 (Рис. 7.8).

Работа с DMA.

Контроллер DMA играет роль канала ввода-вывода, который позволяет разгрузить микропроцессор. Посредством этого контроллера данные из периферийного устройства передаются непосредственно в память (или наоборот).

В машинах класса AT установлено два контроллера DMA, каждый из которых имеет 4 канала. Каждый канал может обслуживать свое устройство. Поскольку один контроллер DMA подключен к одному каналу другого контроллера, то всего получается 7 свободных каналов. Каждый канал имеет свой приоритет, наибольшим приоритетом обладает канал с номером 0.

В контроллере DMA используются 8- и 16-разрядные регистры. Запись в 16-разрядный регистр осуществляется по 8 бит. Вначале пишется младший байт, затем старший. Перед записью следует обнулить специальный указатель (указатель последовательности байт).

В состав контроллера DMA входит 7 регистров:

1. Регистр команд. Адрес 08H. Режим доступа - для записи. Число разрядов - 8. Управляет работой контроллера. Содержимое регистра устанавливается во время инициализации компьютера. Значение битов:

0 - 0 - обмен запрещен, 1 - обмен разрешен.

1 - 0 - фиксация адреса запрещена, 1 - фиксация адреса разрешена.

2 - 0 - контроллер доступен, 1 - контроллер недоступен.

3 - 0 - нормальная передача, 1 - ускоренная передача.

4 - 0 - фиксированные приоритеты, 1 - циклические приоритеты.

5 - 0 - задержка при записи, 1 - расширенная запись.

6 - 0 - запрос активен при высоком уровне сигнала, 1 - запрос активен при низком уровне сигнала.

2. Регистр состояния. Адрес 08Н. Для чтения. Число разрядов - 8.

Указывает состояние линий каналов.

Биты 0-3 - указывают состояние завершения для каналов 0-3. 0 - обмен не завершен, 1 - обмен завершен. Биты 4-7 - определяют состояние линий запроса на обмен для каналов 0-3. 0 - нет запроса, 1 - запрос активен.

3. Регистр запросов. Адрес 09Н. Для записи. Число разрядов - 8. Предназначен для программного указания запроса на работу канала.

Биты 0-1 служат для указания канала:

00 - канал 0,

01 - канал 1,

10 - канал 2,

11 - канал 3.

Бит 2 - устанавливает запрос: 0 - сбросить запрос, 1 - установить запрос.

Остальные биты зарезервированы и могут принимать любое значение.

4. Регистр индивидуальных масок. Адрес 0АН. Для записи. Число разрядов - 8.

Предназначен для маскирования каналов.

Биты 0-1 служат для указания каналов (аналогично регистру запросов).

Бит 2 - команда маскировки: 0 - размаскировать канал, 1 - замаскировать канал.

Остальные биты, зарезервированы и могут принимать любое значение.

5. Регистр режима. Адрес ОВН. Для записи. Число разрядов - 8.

Служит для установки режима работы канала.

Биты 0-1 указывают канал (см, пред.).

Биты 2-3 - тип передачи данных:

00 - блокировка записи в память,

01 - запись в память,

10 - чтение из памяти,

11 - недопустимое значение.

Бит 4 - режим автоинициализации: 0 - автоинициализация выключена, 1 - автоинициализация включена.

Бит 5 - направление изменения адреса при обмене: 0 - увеличение адреса, 1 - уменьшение адреса.

Биты 6-7 - режим передачи данных:

00 - передача по требованию,

01 - передача байта,

10 - передача блока,

11 - каскадирование.

6. Вспомогательный регистр. Адрес ОСН. Для записи. Число разрядов - 8. При записи туда любого числа очищается регистр защелки. Это делается перед установкой режима работы.

7. Регистр масок. Адрес OFH. Для записи. Число разрядов - 8.

Предназначен для управления масками каналов.

Биты 0-3 - определяют каналы: 0 - размаскировать канал, 1 - замаскировать канал.

Остальные биты зарезервированы и могут принимать любое значение.

8, Временный регистр. Адрес 0DH. Для чтения. Число разрядов - 8. Хранит байт для передачи и используется контроллером.

Кроме того, каждый канал имеет по пять регистров:

1. Регистр базового адреса. Разрядность - 16. Для записи.

каналы	адрес
0	00h
1	02h
2	04h
3	06h

2. Регистр текущего адреса. Разрядность - 16. Для чтения.

каналы	адрес
0	00h
1	02h
2	04h
3	06h

3. Регистр базового счетчика. Разрядность - 16. Для записи.

каналы	адрес
0	01h
1	03h
2	05h
3	07h

4. Регистр текущего счетчика. Разрядность 16. Для чтения.

каналы	адрес
0	Olh
1	03h
2	05h
3	07h

5. Регистр страницы. Разрядность 8. Для записи.

0	87h
1	83h
2	81h
3	82h

Адрес памяти для обмена между DMA и периферийными устройствами задается для каждого канала через два регистра: регистр адреса и регистр страницы. В результате имеем 24-разрядный адрес памяти. Границы страницы расположены по адресам 10000H, 20000H и т.д. Поскольку адрес не должен пересекать границу страницы, то передача т.о. может производиться блоками размером не более 64K. Для некоторых современных контроллеров такого ограничения не существует.

При программировании DMA записываются регистры базового адреса, регистр страницы и базовый счетчик.

Режимы работы:

- передача байта по каждому запросу от периферийного устройства;
- передача блока, шина захватывается на время передачи блока;

- передача по требованию, периферийное устройство в любой момент может прервать передачу, а затем возобновить с этого же места;
- каскадирование, позволяет подключать несколько контроллеров DMA. Один из каналов ведущего контроллера используется для подключения ведомого.

Типы передачи:
- автоинициализация, после окончания передачи контроллер приходит в исходное состояние;
- циклический сдвиг приоритетов каналов после каждой передачи;
- ускоренная передача, передача без цикла ожидания.

Дополнительные команды.

Установка начального значения указателя последовательности байт. Для этого записывается произвольное значение в регистр по адресу ОСН.

Общий сброс. Записывается произвольное значение в регистр по адресу 0DH.

Размаскирование каналов. Записывается произвольное значение в регистр по адресу ОЕН.

Во всех приведенных случаях должно записываться 16-битное число.

Расширение для машин AT.
Адресация общих регистров каналов 4-7:
Регистр команд - D0H,
Регистр состояния - D0H,
Регистр запросов - D2H,
Регистр индивидуальных масок - D4H,
Регистр режима - D6H,
Временный регистр - DAH,
Регистр масок - DEH,
Указатель последовательности байт - D8H,
Общий сброс - DAH,
Размаскирование каналов - DCH.

Адресация регистров каналов 4-7:
1. Регистр базового адреса. Разрядность - 16. Для записи.

каналы	адрес
4	C0h
5	C4h
6	C8h
7	CCh

2. Регистр текущего адреса. Разрядность - 16. Для чтения.

каналы	адрес
4	COh
5	C4h

6 C8h
7 CCh

3. Регистр базового счетчика. Разрядность - 16. Для записи.

каналы	адрес
4	C2h
5	C6h
6	CAh
7	CEh

4. Регистр текущего счетчика. Разрядность - 16. Для чтения.

каналы	адрес
4	C2h
5	C6h
6	CAh
7	CEh

5. Регистр страницы. Разрядность - 8. Для записи.

каналы	адрес
4	8Fh
5	8Bh
6	89h
7	8Ah

Программирование DMA.
1. Замаскировать канал.
2. Загрузить регистры адреса, страницы и счетчика.
3. Установить режим работы канала в регистре режима.
4. Размаскировать канал.

Адреса всех указанных регистров совпадают с адресами портов. Так, например, для маскирования канала 3 необходима следующая последовательность команд:

```
MOVAL,00000111B
OUT 0AH,AL
```

Приведем еще несколько полезных фрагментов.
Сбросить запрос для канала 1:

```
MOV AL,00000001B
OUT 09H,AL
```

Для второго канала установить режим автоинициализации с блокированием записи в памяти. Режим - передача блока, направление изменения адреса - уменьшение.

```
MOV AL,0111D010B
OUT OBH,AL
```

Назначение каналов.

0 - свободен,
1 - свободен,
2 - контроллер гибких дисков,
3 - контроллер винчестера,
4 - каскадирование,
5 - свободен,
6 - свободен,
7 - свободен.

Пример программирования DMA приведен ниже (см. контроллер гибких дисков).

Работа с гибкими дисками.

Накопители на гибких дисках соединены с контроллером линиями. Состояние следующих линий отображается в регистрах контроллера:

- сигнал обнаружение индекса,
- сигнал положения головок на дорожке,
- сигнал защиты записи,
- сигнал смены дискеты.

Способ записи на гибкий диск MFM - модифицированная фазовая модуляция.

Для дискет IBM PC существует стандартный формат дорожки. Дорожка состоит из записей следующих типов:

- начало дорожки,
- сектор на дорожке,
- конец дорожки.

Начало дорожки:

- GAP4A - промежуток, записываемый контроллером при форматировании дорожки. 80 байт, заполненных значением 4EH.
- SYNC - промежуток, записываемый контроллером при форматировании дорожки. Содержит 12 байт, заполненных значением 00h.
- IAM - адресный маркер, записываемый при форматировании. Содержит 3 байта C2H со специальным нарушением последовательности бит синхронизации и байт со значением FCH.
- GAP1 - промежуток, записываемый при форматировании дорожки. Содержит 50 байтов, заполненных значением 4EH.

Сектор:

- SYNC - содержит 12 байтов, заполненных значением OOh.
- IDAM - адресный маркер идентификатора сектора. Содержит 3 байта A1H со специальным нарушением последовательности бит синхронизации, а также значение FEh - признак маркера идентификатора.
- CYL - номер цилиндра (один байт).

- HEAD - номер головки. Занимает один байт.
- SEC - номер сектора, один байт.
- NO - код размера сектора, один байт. Длина сектора равна 128*2^NO.
- CRC1 - контрольная сумма идентификатора сектора - два байта.
- GAP2 - промежуток в 22 байта, заполненный 4ЕН.
- DATA AM - адресный маркер данных. Записывается при форматировании и при записи. Содержит 3 байта A1H со специальным нарушением последовательности бит синхронизации и значение FBH (обычные данные), F8H (стертые данные).
- поле данных.
- CRC2 - контрольная сумма данных (2 байта).
- GAP3 - промежуток, записываемый при форматировании. Длина промежутка задается в команде форматирования. Значения:

```
5.25/360 - 80,
5.25/1.2 - 84,
3.5/720 - 80,
3.5/1.4 - 108.
```

Конец дорожки:

- GAP4B - промежуток, записываемый контроллером при форматировании дорожки. Содержит 4ЕН.

Регистры контроллера гибких дисков.

Базовый адрес портов регистров контроллера гибких дисков равен 3F0H.

Регистр состояния. Относительный адрес 4. Предназначен для определения разрешения и направления передачи данных между микропроцессором и контроллером. Доступен по чтению.

Биты 0-1 - устанавливаются в 1 для накопителей 0 и 1 соответственно, когда накопитель находится в режиме установки головки.

Биты 2-3 - резерв.

Бит 4 - устанавливается в 1 сразу после передачи в контроллер байта команды. Сбрасывается после завершения фазы чтения результата.

Бит 5 - устанавливается в 1 во время фазы выполнения команды. Действителен когда в команду SPECIFY указан режим работы без DMA.

Бит 6 - 0 - данные передаются от микропроцессора к контроллеру, 1 - наоборот.

Бит 7 - 0 - передача данных разрешена, 1 - передача данных запрещена.

Регистр данных. Относительный адрес 5. Через этот регистр передается байт данных между микропроцессором и контроллером. Доступен по чтению и записи.

Регистр сигналов входов. Относительный адрес 7. Определяет смену дискеты в накопителе. Доступен по чтению.

Бит 7 (единичное значение) указывает, что дискета была сменена.

Регистр конфигурации. Относительный адрес 7. Определяет скорость передачи данных между контроллером и накопителем. Доступен по записи.

Биты 0-1:

00 - 500 Кб/сек

01 - 300 Кб/сек

10 - 250Кб/сек

11 - резерв.

Регистр управления. Относительный адрес 2. Предназначен для управления накопителем. Доступен по записи.

Биты 0 - выбор накопителя для работы.

Бит 1=1.

Бит 2 - 0 - выполнить сброс, 1 - контроллер доступен.

Бит 3 - 0 - работа с DMA и прерываниями запрещена, 1 - разрешена.

Бит 4 - 0 выключить мотор накопителя 0, 1 - включить мотор.

Бит 5 - 0 выключить мотор накопителя 1, 1 - включить мотор.

Биты 6-7 - резерв.

Регистры состояния. Существуют четыре регистра состояния. Байты из этих регистров можно получить после выполнения команд.

Регистр состояния 0.

Биты 0-1:

00 - накопитель О,

01 - накопитель 1,

10 - накопитель 2,

11 - накопитель 3.

Бит 2 - номер головки (стороны), для которой выполнялась операция.

Бит 3 - резерв.

Бит 4 - устанавливается в 1, если при выполнения команды «рекалибровка», после 77 сигналов управления перемещением головок не получен сигнал "Дорожка О".

Бит 5 - определяет завершение команды установки (1).

Бит 6-7:

00 - нормальное завершение команды,

01 - ненормальное завершение команды,

10 - недействительная команда,

11 - аварийное завершение.

Регистр состояния 1.

Бит 0 - 1 - не найден адресный маркер.

Бит 1 - 1 - если активен сигнал защита записи.

Бит 2 - 1 - не найден сектор или адресный маркер не может быть прочитан.

Бит 3 - зарезервирован.

Бит 4 - 1 - если запрос контроллера не обслужен со стороны процессора или DMA.

Бит 5 - указывает на ошибку контрольной суммы.

Бит 6 - резерв.

Бит 7 - устанавливается в 1 при попытке получить доступ сектору за последним сектором на цилиндре.

Регистр состояния 2.

Бит 0 - 1, если во время операции чтения не найден адресный маркер.

Бит 1 - 1, если внутренний номер цилиндра контроллера не совпал с номером цилиндра из идентификатора сектора и равен FFH.

Бит 2 - резерв.

Бит 3 - резерв.

Бит 4 - 1, если внутренний номер цилиндра контроллера не совпал с номером цилиндра из идентификатора сектора.

Бит 5 - 1 при фиксации ошибки контрольной суммы в поле данных.

Бит 6 - 1 при Попытке получить доступ к сектору с отметкой "стертый".

Бит 7 - резерв.

Регистр состояния 3.

Бит 0 - номер выбранного накопителя.

Бит 1 - резерв.

Бит 2 - номер выбранной головки (стороны).

Бит 3 - резерв.

Бит 4 - 1 когда головки находятся на дорожке 0.

Бит 5 - резерв.

Бит 6 - 1 - установлена защита записи.

Бит 7 - резерв.

Описание команд контроллера.

В общем случае команды контроллера гибких дисков состоят из трех фаз: командная фаза (1), фаза выполнения (2), фаза получения результата (3). Для описания команд введем следующие обозначения, широко используемые в технической литературе по контроллерам гибких дисков.

C - номер цилиндра,

H - номер головки,

R - номер сектора,

N - код длины сектора,

EOT - номер последнего сектора на дорожке,

GPL - длина межсекторного промежутка (GAP3),

DTL - длина передаваемых данных (0 для 128 байтного сектора и FFH в остальных случаях),

STO - регистр состояния O,

ST1 - регистр состояния 1,

ST2 - регистр состояния 2,

ST3 - регистр состояния 3,

HDS - выбор номера головки,

DS - выбор номера накопителя,

HLT - время загрузки головок,

HUT - время разгрузки головок,

MT - многодорожечная операция,

NCN - новый номер цилиндра,
PCN - текущий номер цилиндра,
SK - признак пропуска стертых данных,
SRT - время между последовательными сигналами на перемещение головок,
TC - конец счетчика,
SC - секторов на дорожке,
D - заполнитель,
SSEL - скорость передачи данных между накопителем и контроллером.

Команда чтения данных.

фазы	данные
1	MT 1 SK 0 0 1 1 0
	0 0 0 0 0 HDS 0 DS
	C
	H
	R
	N
	EOT
	GPL
	DTL
2	
	ST0
3	ST1
	ST2
	C
	H
	R
	N

Команда чтения стертых данных.

фазы	данные
1	MT 1 SK 0 1 1 0 0
	0 0 0 0 0 HDS 0 DS
	C
	H
	R
	N
	EOT
	GPL
	DTL

2	
3	STO
	ST1
	ST2
	C
	H
	R
	N

Команда записи данных.

фазы	Данные
1	MT 1 0 0 0 1 0 1
	0 0 0 0 0 HDS 0 DS
	C
	H
	R
	N
	EOT
	GPL
	DTL
2	
3	STO
	ST1
	ST2
	C
	H
	R
	N

Команда записи стертых данных.

фазы	данные
1	MT 1 0 0 1 0 0 1
	0 0 0 0 0 HDS 0 DS
	C
	H
	R
	N
	EOT
	GPL
	DTL

2	
3	STO
	ST1
	ST2
	C
	H
	R
	N

Команда чтения дорожки.

фазы	данные
1	0 1 0 0 0 0 1 0
	0 0 0 0 0 HDS O DS
	C
	H
	R
	N
	EOT
	GPL
	DTL
2	
3	STO
	ST1
	ST2
	C
	H
	R
	N

Команда форматирования дорожки.

фазы	данные
1	0 1 0 0 1 1 0 1
	0 0 0 0 0 HDS 0 DS
	N
	SC
	GPL
	D
	C
	H
	R
	N

2	
3	STO
	ST1
	ST2
	байт неопределен
	байт неопределен
	байт неопределен
	байт неопределен

Команда чтения идентификатора. По этой команде считывается первый обнаруженный на дорожке идентификатор сектора.

фазы	данные
1	0 1 0 0 1 0 1 0
	0 0 0 0 0 HDS 0 DS
2	
3	STO
	ST1
	ST2
	C
	H
	R
	N

Команда рекалибровки. Предназначена для установки начального положения головок на нулевой цилиндр.

фазы	Данные
1	0 1 0 0 0 1 1 1
	0 0 0 0 0 0 0 DS

Команда установки. Предназначена для установки головок на требуемый цилиндр.

Фазы	данные
1	0 0 0 0 1 1 1 1
	0 0 0 0 0HDSODS

Получить результат прерывания.

фазы	данные
1	0 0 0 0 1 0 0 0
3	STO
	PCN

Получить состояние устройства.

фазы	данные
1	0 0 0 0 0 1 0 0
	0 0 0 0 0 HDS O DS
3	ST3

Команда "определить". Предназначена для программирования внутренних таймеров **контроллера.**

фазы	Данные
1	0 0 0 0 0 0 1 1
	--- SRT-- - - HUT---
	--- HLT__ -- ---_- ND

Значения SRT.	Скорость 500K	Скорость 300K	Скорость 250K
0	16	26.7	32
1	15	25	30
...	...	...	...
0EH	2	3.33	4
0FH	1	1.67	2
Значения HUT.	**Скорость 500K**	**Скорость 300K**	**Скорость 250K**
0	256	426	512
1	16	26.7	32
...	...	...	...
OEH	224	373	448
0FH	240	400	480
Значения HLT.	**Скорость 500K**	**Скорость 300K**	**Скорость 250K**
0	256	426	512
1	2	3.3	4
2	4	6.7	8
...	...	...	...
OEH	252	420	504
0FH	254	423	508

Рекомендуемые параметры команд.

Тип носит.	Тип диска	SRT+HUT	HLT	SSEL	GPL R/W|F
5.25/360	360K	DFH	02H	02H	2AH|50H
5.25/1.2	360K	DFH	02H	01H	2AH|50H

5.25/1.2	1.2M	DFH	02H	00H	1BH|54H
3.5/720	720K	DFH	02H	02H	2AH|50H
3.5/1.44	720K	DFH	02H	02H	2AH|50H
3.5/1.44	1.44M	AFH	02H	00H	1BH|6CH

Ниже приведена программа работы с контроллером гибких дисков. Программа читает загрузочный сектор дискеты 1.4. В программе приведены все основные операции, кроме записи и форматирования. Попробуйте осуществить эти операции самостоятельно. Обращаю Ваше внимание на процедуру задержки, которая используется в данной программе, она взята из программы **P12.5.**

```
;чтение данных из нулевого сектора дисковода 1.4
CODE SEGMENT
	ASSUME CS:CODE, DS:CODE
	ORG 100H
BEG:
;координаты для DMA
	CALL SETCOORD
;установка прерывания
	XOR AX,AX
	MOV ES,AX
	MOV AX,ES:[0EH*4+2]
	MOV CS:OLDINTSE,AX
	MOV AX,ES:[0EH*4]
	MOV CS:OLDINTOF,AX
	MOV AX,OFFSET NEWINT
	CLI
	MOV ES:[0EH*4],AX
	MOV AX,CS
	MOV ES: [0EH*4+2],AX
	STI
;установки контроллера
;скорость передачи данных
	MOV DX,03F7H	;адрес регистра скорости передачи данных
	MOV AL,0	;максимальная скорость 500 килобит/сек.
	OUT DX,AL
	JMP SHORT $+2
;сброс контроллера
	CLI
;разрешить DMA, сброс контроллера, выключить мотор
;выключить мотор устройства 0
	MOV AL,00011000B
	MOV DX,03F2H
```

```
        OUT  DX,AL
        JMP  SHORT  $+2
;задержка приблизительно 29 мкс
        MOV  DX,35
        CALL  TIMERM
/разрешить DMA, работа контроллера, включить мотор, устройство О
        MOVAL,00011100B
        MOV  DX,03F2H
        OUT  DX,AL
        JMP  SHORT  $+2
        CALL  WAITINT       ;ждем прерывания от контроллера
        CMP   ERROR,0
        JZ    DAL1
        JMP  _ERR
DALl:
;снимем прерывание для 4 устройств
;четырехкратное снятие необходимо только после команды "СБРОС"
        MOV CX,4
        MOV     DL,11000000B
RES:
        CALL  SENSI
        CMP   ERROR, 1
        JNZ    DAL2
        JMP   _ERR
DAL2:
        MOV   AH,ST0
        TEST  AH,DL
        JZ    DAL3
        JMP   _ERR
DAL3:
        INC   DL  ;следующее устройство
        LOOP  RES
;ждем раскрутки мотора  (0.5 сек)
;возможно, что для некоторых устройств это время придется увеличить
        MOV  CX,500
_WAIT1:
        MOV  DX,1190
        CALL  TIMERM
        LOOP  _WAIT1
;выполним команду  SPECIFY
;необходима для определения внутренних
;временных интервалов контроллера
        MOV   AL,3
        CALL  OUTFP
        CMP   ERROR, 1
```

```
        JNZ   DAL4
        JMP  _ERR
DAL4:
        MOV   AL,0AFH ;для 1.4
        CALL  OUTFP
        CMP   ERROR,1
        JNZ   DAL5
        JMP    _ERR
DAL5:
        MOV   AL,2H    ;для всех
        CALL  OUTFP
        CMP   ERROR,1
        JNZ   DAL6
        JMP    _ERR
DAL6:
;команда RECALIBRATE
;установка начального положения головок
;команда выполняется дважды, т.к. для 80 дорожек
;одной может не хватить
        MOV  CX, 2
RECAL:
        CLI
        MOV AL,7
        CALL OUTFP
        CMP   ERROR,1
        JNZ   DAL7
        JMP    _ERR
DAL7:
        MOV  AL, 0   ;устройство 0
        CALL  OUTFP
        CMP   ERROR,1
        JNZ   DAL8
        JMP    _ERR
DAL8:
        CALL  WAITINT
        CMP   ERROR,1
        JNZ  DAL9
        JMP    _ERR
DAL9:
;снять прерывание
        CALL  SENSI
        CMP   ERROR,1
        JNZ   DAL10
        JMP    _ERR
DAL10:
```

```
;проверим результат
        MOV   AH,ST0
        AND   AH,11000000B
        JZ    COMPL
ESHE:
        LOOP  RECAL
        JMP   _ERR
COMPL:
        MOV DX,18750    ;время успокоения головок 15 мс
        CALL TIMERM
;читать один сектор в начале дискеты
;команда SEEK
;переместить головку к нужной дорожке
        CLI
        MOV   AL,0FH
        CALL  OUTFP
        CMP   ERROR, 1
        JNZ   DAL23
        JMP   _ERR
DAL23:
        MOV  AL, 0
        CALL  OUTFP
        CMP   ERROR,1
        JNZ   DAL231
        JMP   _ERR
DAL231:
        MOV   AL,0         ;нулевая дорожка
        CALL  OUTFP
        CMP   ERROR,1
        JNZ   DAL232
        JMP   _ERR
DAL232:
        CALL  WAITINT        ;ждем прерывания от контроллера
        CMP   ERROR,1
        JNZ    DAL589
        JMP    _ERR
DAL589:
        CALL  SENSI          ;снять  прерывание
        CMP   ERROR,1
        JNZ    DAL345
        JMP   _ERR
DAL345:
        MOV  DX,18750   ;время успокоения головок 15 мс
        CALL  TIMERM
```

```
;установим DMA на передачу данных из контроллера в
;память (команда 46H)
        MOV    CX,512
        MOV    AL,046H
        CALL  SETDMA
;команда чтения
        CLI
        MOV AL,046H          ;команда чтения
        CALL  OUTFP
        CMP   ERROR,1
        JNZ   DAL11
        JMP    _ERR
DAL11:
;-------------------
        MOV   AL,0           ;накопитель 0, поверхность 0
        CALL  OUTFP
        CMP   ERROR,1
        JNZ   DAL111
        JMP _ERR
DAL111:
;-------------------
        MOV  AL, 0           ;цилиндр 0
        CALL  OUTFP
        CMP   ERROR,1
        JNZ    DAL1112
        JMP _ERR
DAL1112:
;-------------------
        MOV AL,0             ;поверхность 0
        CALL  OUTFP
        CMP   ERROR,1
        JZ  _ERR
;-------------------
        MOV   AL,1           ;сектор 1
        CALL  OUTFP
        CMP   ERROR,1
        JZ  _ERR
;       -------------
        MOV   AL,2           ;код длины 2
        CALL  OUTFP
        CMP  ERROR, 1
        JZ  _ERR
;-------------------
        MOV   AL,18          ;количество секторов на дорожке
        CALL  OUTFP
```

```
        CMP   ERROR,1
        JZ    _ERR
;---------------------
        MOV   AL,1BH        ;длина межзонного промежутка
        CALL  OUTFP
        CMP   ERROR,1
        JZ    _ERR
;---------------------
        MOV   AL,0FFH       ;длина данных
        CALL  OUTFP
        CMP   ERROR,1
        JZ    _ERR
;---------------------
        CALL  WAITINT       ;ждем прерывания от контроллера
        CMP   ERROR,1
        JZ    _ERR
;результат
        CALL  SENSI         ;снять прерывание
        CMP   ERROR,1
        JZ    _ERR
        MOV   AH,ST0
        AND   AH,11000000B
        JNZ   _ERR
        CALL  WSE
;       вывести результат
        LEA  DX,TEXT2
        MOV  AH,9
        INT  21H
;в нулевом (загрузочном) секторе по смещению 36H
;находится слово (8 символов), определяющее тип
;носителя. Обычно это FAT12.
        MOV  SI,OFSSECT
        ADD  SI,36H
        MOV  CX,8
L002:
        LODSB
        MOV  DL,AL
        MOV  AH,2
        INT  21H
        LOOP L002
        JMP  SHORT _END
;конец работы
_ERR:
```

```
;вывод содержимого регистра STO (как символа)
      MOV DL,STO
      MOV AH,2
      INT 21H
      MOV AH,9
      LEA DX,TEXT1
      INT 21H
_END:
      CALL WSE
      RET
;область процедур
;установить DMA на передачу данных
;AL - код операции
;CX - длина данных
SETDMA      PROC
      PUSH    AX
      PUSH    CX
      PUSH    AX
;маскируем второй (10) канал
      CLI
      MOV     AL,00000110B
      OUT     0AH,AL
      JMP     $+2
      POP     AX
      OUT     0CH,AL      ;очистка регистра защелки
      JMP     $+2
;в регистр режима помещаем код операции
;в нашем примере это 46H = 01000110B
;т.е. канал 2 (10), чтение в память (01)
;автоинициализация выключена (0), адрес увеличивается (0),
;передача байта (01)
      OUT     0BH,AL
      JMP     $+2
;запомнить количество передаваемых байт
      PUSH    CX
;теперь адрес памяти в DMA
      MOV     AX,OFSDMA   ;смещение
      OUT     4,AL        ;младший байт
      JMP     $+2
      MOV     AL,AH
      OUT     4,AL        ;старший байт
      JMP     $+2
      MOV     AL,SEGDMA   ; номер страницы
      OUT     81H,AL
      JMP     $+2
```

```
;теперь установим счетчик
;он должен содержать 512-1
        POP      AX
        DEC      AX
        OUT      5,AL      ;младший байт
        JMP      $+2
        MOV      AL,AH
        OUT      5,AL      ;старший байт
        JMP      $+2
;размаскировать второй канал
        MOV      AL,00000010B
        OUT      0AH,AL
        JMP      $+2
        STI
        POP      CX
        POP      AX
        RET
SETDMA       ENDP
;процедура получения данных от контроллера
INFP  PROC
        PUSH  AX
        PUSH  BX
        PUSH  CX
        PUSH  DX
        PUSH  DI
        MOV   DI,OFFSET ST0
        MOV   ERROR,0
        MOV   DX,3F4H          ;регистр состояния
        MOV   BH,7
L0:
        MOV  BL,2
        MOV  CX,0
L1:
;проверка разрешения передачи данных
        IN  AL,DX
        TEST  AL,10000000B
        JNZ   L2
        LOOP  L1
        DEC   BL
        JNZ   L1
        MOV   ERROR,1
        JMP   SHORT  ER
L2:
        INC  DX            ;адрес регистра данных
        IN AL,DX
```

```
        STOSB            ;байт в ячейку STO
        DEC DX           ;адрес регистра состояния
        PUSH DX
;небольшая задержка
        MOV  DX,35
        CALL TIMERM
        POP  DX
;проверка конца фазы чтения результата
        IN   AL,DX
        TEST AL,00010000B
        JZ   ER
        DEC  BH
        JNZ  LO          ;повторить чтение
        MOV  ERROR,1
ER:
        POP  DI
        POP  DX
        POP  CX
        POP  BX
        POP  AX
        RET
INFP ENDP
;процедура передачи данных в контроллер
/AL - байт данных
OUTFP PROC
        MOV ERROR,0
        PUSH AX
        PUSH BX
        PUSH DX
        PUSH CX
        MOV  DX,3F4H     ;регистр состояния
        MOV  BL,2
        XOR CX,CX
        MOV AH,AL
LOO1:
;проверка, разрешена ли передача данных
        IN AL,DX         /основной регистр состояния контроллера
        TEST AL,10000000B
        JNZ  _OUT
        LOOP  L001
        DEC BL
        JNZ L001
        MOV ERROR,1
        JMP SHORT _END_OUT
_OUT:
```

```
        INC DX              ;адрес регистра передачи данных
        MOV AL,AH
        OUT DX,AL           ;передали байт
        JMP SHORT $+2
;задержка 29 мк и выход
        MOV DX,35
        CALL TIMERM
_END_OUT:
        POP CX
        POP DX
        POP BX
        POP AX
        RET
OUTFP ENDP
;задержка N мкс.
; N/(0.840336)=M
;процедура задержки
; параметр в DX (M)
TIMERM PROC
        PUSH AX
        PUSH DX
;отключаем работу канала 2
        IN   AL, 61H
        AND AL,11111100B
        OUT 61H,AL
        JMP SHORT $+2
;управляющий байт:
;двоичный формат
/режим 2
;чтение и запись
;канал 2
        MOV AL,10110100B
        OUT 43H,AL
        MOV AL,0FFH
        OUT 42H,AL
        JMP SHORT $+2
        OUT 42H,AL
        JMP SHORT $+2
        IN AL,61H
        OR AL,01H
/включить счетчик
        OUT 61H,AL
        JMP SHORT $+2
```

```
;получить значение счетчика, когда
;следует остановиться
      NEG DX
WAITING:
;управляющий байт
;режим 2
;"защелкнуть" счетчик
      MOV AL,10000100B
      OUT 43H,AL
      JMP SHORT $+2
      IN AL,42H
      XCHG AH,AL
      IN AL,42H
      XCHG AH,AL
      CMP DX,AX
      JB WAITING
      POP DX
      POP AX
      RETN
TIMERM ENDP
;процедура прерывания от контроллера гибких дисков
NEWINT PROC
      PUSH AX
      MOV FLAG,1   ;установили флаг
;стандартная команда завершения прерывания
      MOV AL,20H
      OUT 20H,AL
      POP AX
      IRET
NEWINT ENDP
;команда SENSI - снять прерывание
SENSI PROC
      PUSH AX
      MOV  AL, 8       ; код команды
      CALL OUTFP
      CMP  ERROR,1
      JZ   _ENDS
      CALL INFP        ;читаем STO
_ENDS:
      POP AX
      RET
SENSI ENDP
;процедура ожидания прерывания
WAITINT PROC
      PUSH CX
      PUSH DX
```

```
        MOV  FLAG,0
        MOV  ERROR,0
        STI
;ждем две  секунды
        MOV  CX,2000
_WAIT:
        MOV  DX,1190
        CALL  TIMERM
        CMP  FLAG,1      ;флаг взведен
        JZ   _RET
        LOOP  _WAIT
        MOV   ERROR, 1
_RET:
        POP  DX
        POP  CX
        RET
WAITINT  ENDP
;процедура определения  координат для  DMA
SETCOORD  PROC
;вначале анализ  того,  не  переходит ли  буфер  для
;данных через  границу  страницы
        LEA      AX,SECTOR
        MOV      BX,CS
        MOV      CL,4
        SHL      BX,CL
        ADD      BX,AX   ;суммарное смещение  в  странице
        JC       DAL
        MOV      DX,BX
;не перешла  ли   через  границу  полезная  часть  буфера
        ADD      BX,512
        JNC      DAL
;новое смещение  в  сегменте  для  прочитанных  данных
;смещение же  в  странице  в  результате  окажется  равным  0
        SUB      AX,DX
DAL:
        MOV      OFSSECT,AX ; смещение  данных
;определить  адреса  для  DMA
        MOV      DX,AX
        MOV      AX,CS
        MOV      CL,4
;процесс отделения  адреса  страницы  от  смещения  в  странице
        ROL      AX,CL
        MOV      CH,AL        ;биты для  адреса  страницы
```

```
        AND     AL,0F0H     ;оставим только смещение от сегмента
        ADD     AX,DX       ;смещение в странице
        ADC     CH,0        ;учет перехода через границу
        MOV     OFSDMA,AX
        AND     CH,0FH      ;работают только старшие 4 бита
        MOV     SEGDMA,CH
        RET
SETCOORD ENDP
;процедура завершения
WSE PROC
;выключаем мотор
;разрешить DMA, работа контроллера, выключить мотор, устройство О
        MOV AL,00001000B
        MOV DX,03F2H
        OUT DX,AL
        JMP SHORT $+2
;восстановим вектор
        MOV AX,OLDINTOF
        CLI
        MOV ES:[0EH*4],AX
        MOV AX,OLDINTSE
        MOV ES:[0EH*4+2],AX
        STI
        RET
WSE ENDP
;размер буфера 512 байт на случай, если буфер перейдет
/через границу страницы DMA
SECTOR  DB      2*512 DUP(0) /буфер для чтения сектора
;смещение в сегменте прочитанных данных из сектора
OFSSECT   DW    ?
OFSDMA    DW    ?
SEGDMA    DB    ?
ERROR     DB    ?   /признак ошибки
OLDINTOF DW ?
OLDINTSE DW ?
FLAG DB 0       /флаг выполнения команд контроллера
/область для получения данных от контроллера
STO  DB 0       /здесь будем помещать содержимое регистра STO
TEXT1   DB 'Ошибка',13,10, '$'
TEXT2   DB 'Тип носителя ','$'
CODE ENDS
        END BEG
```

Рис. P12.2. Чтение загрузочного сектора дискеты.

Работа с видеосистемой.

Справочный материал помещен в главу 27.

Порты для работы с жесткими дисками.

Жесткие диски разных марок и разных выпускающих фирм в значительной степени отличаются друг от друга. Поэтому нет смысла рассматривать здесь программирование ЖД на низком уровне. На практике для этой цели всегда используются средства BIOS.

Работа с часами реального времени.

Часы появились в компьютерах AT, дабы ликвидировать неудобства, связанные с тем, что при выключении компьютера обнулялись и системные часы. Теперь при запуске операционная система считывает время с часов реального времени.

Часы позволяют генерировать три типа прерываний:

- периодическое прерывание с интервалом 976.562 мкс.;
- прерывание от будильника;
- прерывание по окончанию обновления значения часов.

Прерывание происходит по линии 0 второго контроллера прерываний (IRQ 8).

Часы используют 14 байт полупостоянной памяти. Доступ к этой памяти осуществляется через порты 70H, 71H (см. главу 23). Ниже мы приводим назначение 14 байт часов реального времени (RTC):

Байт	Назначение
OOH	Секунды
01H	Секунды для будильника
02H	Минуты
O3H	Минуты для будильника
04H	Часы
05H	Часы для будильника
06H	День недели
07H	Дата
08H	Месяц
09H	Год
0AH	Регистр состояния 1
OBH	Регистр состояния 2
0CH	Регистр состояния 3
ODH	Регистр состояния 4
32H	Столетие

Более подробные сведения о содержимом CMOS Вы можете найти в главе 23, здесь же сконцентрируемся на программировании самих часов.

1. Установка-чтение времени. Чтение часов можно производить, только если 7-й бит регистра состояния 2 равен 0. Установку же времени и даты следует производить, запретив работу часов (установить в 1 7-бит регистра состояния 3).

2. Для получения прерываний от будильника следует разрешить это прерывание (регистр состояния 2, бит 5). При установке будильника следует ждать, когда бит 7 регистра состояния 2 равен 0.

3. Для разрешения периодического прерывания через 976.562 мкс. необходимо установить бит 6 регистра состояния 2.

4. Чтобы разрешить его, следует установить бит 4 в регистре состояния 2.

Ниже приводится пример простой программы (Рис. P12.3) на прерывание от часов реального времени. Программа устанавливает вектор прерывания 70H на свою процедуру. В процедуре, в свою очередь, стоит команда инкремента счетчика. Как только счетчик достигает определенного значения, происходит выход из программы. Хочу заметить, что процедура будет выполняться не 1024 раза в секунду (1/0.000976562 = 1024), а много меньше. И связано это с тем, что команды, стоящие в процедуре обработки прерывания, выполняются довольно медленно.

```
CODE  SEGMENT
       ASSUME CS:CODE, DS:CODE
       ORG 100H
BEGIN:
       MOV WORD PTR PRIZN,0  ;обнулить счетчик
;установить вектор прерывания
       MOV AX,0
       MOV  ES,AX
;сохранить старый вектор
       MOV  AX,ES:[70H]
       MOV  OFF_70,AX
       MOV  AX,ES:[70H+2]
       MOV  SEG_70,AX
/установить новый
       CLI
       LEA  AX,INT_70
       MOV  ES:[70H],AX
       PUSH DS
       POP   AX
       MOV   ES:[70H+2],AX
/разрешить прерывание каждые 976.562 мкс.
       MOV AL,OBH
       OUT  70H,AL
       JMP  SHORT  $+2
       IN   AL,71H
       OR   AL,01000000B
       MOV  AH,AL
       MOV  AL,0BH
```

```
        OUT 70H,AL
        MOV AL,AH
        JMP  SHORT  $+2
        OUT 71H,AL
        STI
;цикл ожидания
LOO:
        CMP WORD PTR PRIZN,100
        JB  LOO
;запретить прерывание
        MOV AL,OBH
        OUT 70H,AL
        JMP  SHORT  $+2
        IN  AL,71H
        AND AL,10111111B
        MOV AH,AL
        MOV AL,0BH
        OUT 70H,AL
        MOV AL,AH
        JMP  SHORT  $+2
        OUT 71H,AL
;восстановить старый вектор
        MOV AX,OFF_70
        MOV ES:[70H],AX
        MOV AX,SEG_70
        MOV ES:[70H+2],AX
        RET
;процедура прерывания
INT_70 PROC
;увеличиваем счетчик
        INC WORD PTR CS:PRIZN ;нельзя DS:PRIZN
;разрешаем следующее прерывание
        MOV  AL,OCH
        OUT  70H,AL
        JMP  SHORT  $+2
       . IN AL,71H
        IRET
INT_70 ENDP
;данные
PRIZN DW ?
OFF_70 DW ?
SEG_70 DW ?
CODE ENDS
        END BEGIN
```

Рис. P12.3.

Прерывание 6т часов реального времени.

Обращаю Ваше внимание на следующую команду: INC WORD PTR CS:PRIZN. Нельзя писать INC WORD PTR DS:PRIZN, т.к. DS во время прерывания может указывать куда угодно (см. также главу 9).

Работа с последовательным интерфейсом.

Контроллер последовательного интерфейса предназначен для обеспечения связи по протоколу RS232C.

Состав контроллера:

регистры буфера приемника и передатчика,

регистры разрешения и идентификации прерываний,

регистры управления и состояния линии,

регистры управления и состояния модема,

регистры буфера делителя генератора.

Адреса портов последовательного интерфейса отсчитываются относительно базовых адресов. Базовые же адреса хранятся:

COM1 - 0040H:0000

COM2 - 0040H:0002

COM3 - 0040H:0004

COM4 - 0040H:0006

Регистры:

Регистр управления линией. Адрес 3 относительно базового адреса. Доступен по чтению и записи.

Биты 0-1 - длина слова обмена:

00 - 5 битов,

01 - 6 битов,

10-7битов,

11 - 8 битов.

Бит 2 - определяет количество стоп-битов, 0 - 1 стоп-бит, 1 - 2 - стоп-бита (1.5 при длине слова 5 бит).

Бит 3 - 1 - генерируется бит контроля четности между последним битом и стоп-битом.

Бит 4 - режим контроля четности (0 - четное, 1 - нечетное).

Бит 5 - режим неизменности бита контроля четности.

Бит 6 - 1 - устанавливается состояние "пауза", которое может быть изменено только переустановкой этого бита.

Бит 7 - определяет доступ к некоторым другим регистрам (см. ниже).

Регистр буфера передатчика. Имеет адрес 0 относительно базового адреса. Доступен по записи. Бит разрешения должен быть равен 0.

Регистр буфера приемника. Имеет адрес 0 относительно базового адреса. Доступен по чтению. Бит разрешения должен быть равен 0.

Регистр буфера младшего байта делителя. Имеет адрес 0 относительно базового адреса контроллера. Доступен по чтению и записи. Бит разрешения должен быть равен 1.

Регистр буфера старшего байта делителя. Имеет адрес 1 относительно базового адреса контроллера. Доступен по чтению и записи. Бит разрешения должен быть равен 1.

Ниже приведена таблица делителей и соответствующих им частот.

Делитель	Частота
0900h	50
0600h	75
0600h	150
0180h	300
00C0h	600
0060h	1200
0040h	1800
0030h	2400
0020h	3600
0018h	4800
0010h	7200
000Ch	9600
0006h	19200
0003h	38400
0002h	57600
0001h	115200

Регистр разрешения прерываний. Имеет адрес 1 относительно базового. Доступен по чтению и записи. Бит разрешения должен быть равен 0. Формат регистра:

бит 0 - прерывание по доступности принимаемых данных, 1 - прерывание вырабатывается, 0 - запрещено,

бит 1 - прерывание при освобождении регистра буфера принимаемых данных, 1 - прерывание вырабатывается, 0 - запрещено,

бит 2 - прерывание при изменении состояния линии приемника, 1 - вырабатывается, 0 - запрещено,

бит 3 - прерывание при изменении состоянии модема, 1 - прерывание вырабатывается, 0 - запрещено,

4-7 - равны нулю.

Регистр идентификации прерывания. Имеет адрес 2 относительно базового адреса. Доступен только по чтению. Формат регистра:

бит 0 - 1 - нет ждущих прерываний, 0 - есть ждущее прерывание,

биты 1-2:

11 - изменилось состояние линии приемника, ошибка переполнения, четности или пауза, условие сброса - читать регистр состояния линии,

10 - принимаемые данные доступны, условие сброса - чтение буфера приемника,

01 - освобожден регистр буфера, условие сброса - чтение регистра прерываний или запись в буфер передатчика,

00 - изменилось состояние модема, условие сброса - чтение регистра состояния модема.

Остальные биты должны быть равны 0.

Регистр управления модемом. Имеет адрес 4 относительно базового адреса. Доступен по чтению и записи.

Бит 0 - 1 - сигнал "готовность терминала" устанавливается равным 1, 0 - в противном случае,

бит 1 - 1 - сигнал "запрос на передачу" устанавливается равным 1,

бит 2 - 1 - сигнал Out1 равен 1,

бит 3 - 1 - сигнал Out2 равен 1,

бит 4 - задает режим шлейфа для диагностических целей.

Регистр состояния линии. Имеет адрес 5 относительно базового адреса. Доступен только по чтению.

бит 0 - 1 - приемник полностью принял символ,

бит 1 - 1 - ошибка переполнения - символ помещается в буфер приема, из которого еще не взят символ,

бит 2 - индикатор ошибки четности,

бит 3 - индикатор ошибки стоп-бита,

бит 4 - индикатор состояния "пауза",

бит 5 - индикатор освобождения буфера передатчика,

бит 6 - индикатор освобождения передатчика.

Биты ошибок сбрасываются при чтении регистра состояния линии.

Регистр состояния модема. Имеет адрес 6 относительно базового адреса. Доступен только по чтению.

бит 0 - индикатор изменения сигнала CTS,

бит 1 - индикатор изменения сигнала DSR,

бит 2 - индикатор заднего фронта сигнала,

бит 3 - индикатор изменения сигнала DCD,

бит 4 - инвертированный сигнал CTS,

бит 5 - инвертированный сигнал DSR,

бит 6 - инвертированный сигнал RI,

бит 7 - инвертированный сигнал DCD.

Примеры программы взаимодействия с COM портом приведен в главе 9 (Рис. 9.4).

Управление таймером.

Схема работы таймера.

Интервальный счетчик работает независимо от процессора. Он содержит три независимых счетчика, служащих для отсчета времени и называемых каналами.

Программная модель работы таймера.

Регистр управляющего байта. Работает на запись. Адрес порта 43Н. В него загружается управляющий байт (команда).

Регистр счетчика канала таймера. Содержит 16 бит. Адреса: канал 0 - 40Н, 1 - 41Н, 2 - 42Н. Загрузка осуществляется побайтно (вначале в старший).

Выходной регистр счетчика. Служит для хранения текущего значения счетчика. Считывание происходит побайтно (вначале младший). Адреса такие же, как у предыдущего регистра.

Внутренний регистр состояния.

Выходной регистр состояния. Служит для чтения состояния таймера. Адреса портов такие же, как у счетчика. Можно читать по команде - чтение состояния.

Входные и выходные сигналы:

а) CLK - сигнал внешнего кварцевого генератора,

б) GATE - сигнал управления каналом таймера, управляет только каналом 2, служит для разрешения или запрете счета и др.,

в) OUT - выходной сигнал таймера: канал 0 - поступает в контроллер на линию прерываний 0, 1 - на контроллер регенерации памяти, 2 - подключен к динамику.

Управляющий байт таймера.

Бит 0 - 0 - счетчик в двоичном формате, 1 - в двоично-десятичном формате.

Биты 1-3 - режимы работы канала таймера:

000 - режим 0,

001 - режим 1,

x10 - режим 2,

x11 - режим 3,

100 - режим 4,

101 - режим 5.

x - любой бит.

Биты 4-5 - режим чтения-записи счетчика:

00 - команда защелкивания выходного счетчика канала таймера,

01 - чтение и запись только младшего байта счетчика,

10 - чтение и запись только старшего байта счетчика,

11 - чтение и запись сначала младшего, а затем старшего байта.

Биты 6-7 - биты выбора канала:

00 - канал 0,

01 - канал 1,

10 - канал 2,

11 - команда чтения состояния.

Инициализация таймера.

Инициализация счетчика состоит из двух шагов:

а) запись управляющего байта, в соответствующем канале сразу прекращается счет.

б) загрузка счетчика.

Чтение состояния таймера.

Вначале засылается байт в управляющий регистр (см. выше).

Формат байта состояния:

бит 7 - 0 - низкий уровень сигнала, 1 - высокий уровень,

бит 6 - 0 выходной регистр счетчика отражает текущее значение, 1 - не отражает, остальные биты аналогичны битам управляющего байта при инициализации таймера.

Чтение значения счетчика.

Чтение значения счетчика можно прочесть из порта 42Н.

Однако здесь может возникнуть проблема недостоверности:

- из-за того, что значение счетчика может быть неопределенным в момент начала нового цикла счета и в момент загрузки управляющего байта,
- значение младшего и старшего байтов счетчика может быть рассогласованным.

Данная проблема может быть решена двумя способами:

- приостановить работу счетчика (только канала 2),
- защелкнуть выходной регистр счетчика (см. команды). Защелка автоматически снимается чтением счетчика и перепрограммированием канала.

Режимы работы таймера.

Здесь дается весьма краткое описание режимов. Для программирования вполне достаточно этого (и моих примеров).

Режим 0 обычно используют для генерации единичного импульса. Счет происходит с переходом через ноль.

Режим 1. Используется для генерации звука (периодически поступающие сигналы). Работает только для канала 2.

Режим 2. Также генерирует периодически поступающие сигналы. Используют для часов реального времени.

Режим 3. Режим может использоваться и для звуковых сигналов, и для часов реального времени. По достижению нуля значение счетчика восстанавливается и счет повторяется.

Режим 4. По достижению 0 счет продолжается с переходом через нуль (аналогично режиму 0).

Режим 5. Работает только для канала 2. Счет происходит с переходом через 0.

Механизм работы таймера.

Кварцевый генератор с периодом примерно 0.840336 мкс. посылает сигнал на таймер. При получении сигнала содержимое счетчиков уменьшается на 1.Т.о. изменение счетчика на 1 происходит через 0.840336 мкс. Максимальное число, которое можно поместить в счетчик, равно OFFFFH=65535. Т.о. полный период таймера, т.е. время уменьшение счетчика до нуля, составляет 0.840336*65535=55071.41976 мкс. Следовательно, количество периодов, соответствующее N с., можно вычислить по формуле (N*1000000)/55071.41976.

Пример программирования таймера. Использование таймера для генерации звука рассмотрено в главе 6.

Программа 1. Ранее мы узнали, как организовывать независимый от скорости выполнения команд таймер. Приведем пример еще одного таймера, работающего путем непрерывного опроса таймера компьютера. Процедура, приведенная в программе, может быть использована для создания своих процедур задержек, например, в языках высокого уровня.

```
CODE  SEGMENT
      ORG  100H
      ASSUME CS:CODE, DS:CODE
BEGIN:
      MOV AH, 9
      LEA  DX,TEXT1
      INT  21H
      MOV  CX,182
;задержка 10 с.
;10/(0.000000840336*65535)=181.58238962
      CALL  TIMER
      MOV AH, 9
      LEA  DX,TEXT2
      INT  21H
      RET
;процедура  задержки
; параметр  в  CX
TIMER  PROC
;отключаем работу  канала  2
      IN AL,61H
      AND  AL,11111100B
      OUT  61H,AL
      JMP  SHORT  $+2
;управляющий байт:
;двоичный формат
;режим 2
;чтение и  запись
;канал 2
      MOV  AL,10110100B
      OUT  43H,AL
      MOV  AL,0FFH
;пишем младший  байт
      OUT  42H,AL
      •JMP  SHORT  $+2
;пишем старший  байт
      OUT  42H,AL
      JMP  SHORT  $+2
```

```
       IN AL,61H
       OR AL,01H
;включить счетчик
       OUT 61H,AL
       JMP SHORT $+2
       MOV BX,0FFFFH
WAITING:
;управляющий байт
;режим 2
;"защелкнуть" счетчик
       MOV AL,10000100B
       OUT 43H,AL
       JMP SHORT $+2
;читаем младший байт
       IN AL,42H
       XCHG AH,AL
;читаем старший байт
       IN AL,42H
       XCHG AH,AL
;проверка прохода счетчика через нуль
       CMP AX,BX
       MOV BX,AX
       JB  WAITING
       LOOP WAITING
       RETN
TIMER ENDP
TEXT1 DB 'Ждем...10 с.  ',13,10,'$'
TEXT2 DB 'Конец... ',13,10,'$'
CODE ENDS
       END BEGIN
```

Рис. ***P12.4.*** *Программа задержки.*

Алгоритм проверки перехода счетчика через нуль достаточно прост, однако укажем на [25], где приводится аналогичный алгоритм.

Еще одна программа, осуществляющая задержку в 10 с., но на базе процедуры TIMERM, которая осуществляет задержку в единицах изменения счетчика. Данная процедура удобна для задержек на короткий интервал времени.

```
CODE  SEGMENT
       ORG 100H
       ASSUME CS:CODE, DS:CODE
BEGIN:
       MOV AH,9
       LEA DX,TEXT1
       INT 21H
```

```
;цикл обеспечит задержку на 10 с.
      MOV CX,10000
LOO:
;задержка 1000 мкс.
; 1000/(0.840336)=1190.0001904
      MOV DX,1190
      CALL TIMERM
      LOOP LOO
      MOV AH,9
      LEA DX,TEXT2
      INT 21H
      RET
;процедура задержки
;параметр в DX
TIMERM PROC
;отключаем работу канала 2
      IN AL,61H
      AND AL,11111100B
      OUT 61H,AL
      JMP SHORT $+2
; управляющий байт:
;двоичный формат
;режим 2
;чтение и запись
;канал 2
      MOV AL,10110100B
      OUT 43H,AL
      MOV AL,OFFH
      OUT 42H,AL
      JMP SHORT $+2
      OUT 42H,AL
      JMP SHORT $+2
      IN AL,61H
      OR AL,01H
;включить счетчик
      OUT 61H,AL
      JMP SHORT $+2
;получить значение счетчика, когда
;следует остановиться
      NEG DX
WAITING:
;управляющий байт
;режим 2
;"защелкнуть" счетчик
```

```
        MOVAL,10000100B
        OUT 43H,AL
        JMP  SHORT  $+2
        IN AL,42H
        XCHG AH,AL
        IN AL,42H
        XCHG AH,AL
        CMP DX,AX
        JB  WAITING
        RETN
TIMERM ENDP
TEXT1 DB 'Ждем...10 с. ',13,10,'$'
TEXT2 DB 'Конец...  ',13,10,'$'
CODE ENDS
        END BEGIN
```

Рис. Р12.5. Еще одна программа задержки.

Контроллер прерываний.

I. Схема функционирования.

В главе 9 мы рассказывали Вам о контроллере прерываний. Здесь мы дадим справочный материал о структуре и программировании этого устройства. Ниже приведен рисунокизглавы9,поясняющийсхемуфункционированияконтроллерапрерываний.

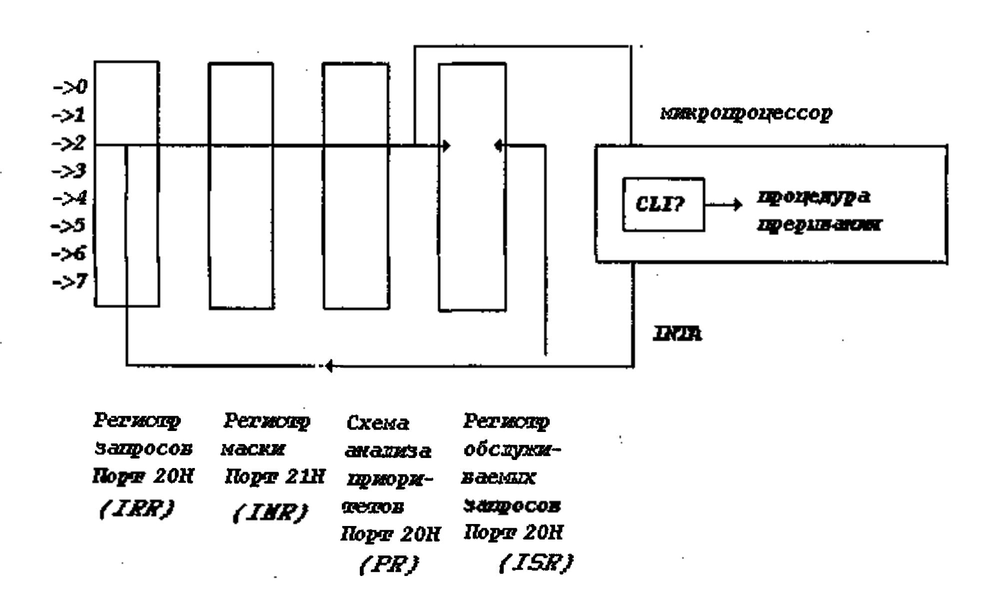


Схема работы прерываний:

1. На одной или нескольких линиях запроса прерываний появляется единичный сигнал. А в регистре IRR устанавливается соответствующий бит.

2. Далее сигнал поступает в регистр масок (IMR). Пройти этот сигнал может, если соответствующий бит в регистре равен нулю.

3. После этого сигнал поступает в регистр анализа приоритетов (PR). Данный регистр срабатывает лишь в том случае, если несколько прерываний поступили одновременно.

4. Далее сигнал одновременно поступает в микропроцессор (сигнал INT) и в регистр обслуживаемых запросов (ISR).

5. Если микропроцессор блокирован командой CLI, то сигнал не пропадает, т.к. остается установленный бит в IRR. После разрешения прерываний сигнал INT будет повторен.

6. Если прерывание разрешено, то из микропроцессора будет послан сигнал INTA. При этом устанавливается бит в регистре ISR и сбрасывается бит в регистре IRR. После этого контроллер может воспринимать сигнал прерывания от того же устройства.

7. Микропроцессор выполняет процедуру прерывания. При этом не явно выполняется команда CLI.

Контроллер прерываний допускает каскадное подключение. В компьютерах AT имеется два контроллера. Сигнал запроса прерывания с одного контроллера (ведомый) поступает не на микропроцессор, а на вход другого (ведущий). Т.о. одновременно может обслуживаться до 15 запросов: 8 на ведомый и 7 на ведущий. Приведем стандартное подключение внешних устройств:

Уровень (приоритет)	Устройство
0	Таймер
1	Клавиатура
2	Ведомый контроллер
8	Часы реального времени
9	Перенаправление линии 2
10	Резерв
11	Резерв
12	Резерв
13	Прерывание сопроцессора
14	Контроллер винчестера
15	Резерв
3	COM2
4	COM 1
5	LPT2
6	Контроллер НГМД
7	LPT1

Режимы работы.

Завершение прерывания.

1. Команда конца прерывания сбрасывает бит наиболее приоритетного прерывания (именно оно и обслуживается в текущий момент).

2. Команда конца прерывания сбрасывает конкретный бит прерывания.

3. Автоматический конец прерывания по приходу импульса INTA.

Приоритеты прерываний.

1. Режим полной вложенности. Все входные линии упорядочены по приоритетам. Обрабатывается всегда линия с наибольшим приоритетом.

2. Специальный режим полной вложенности. Похож на предыдущий с той лишь разницей, что при обслуживании сигнала от ведомого контроллера он не блокируется.

3. Автоматический сдвиг приоритетов. После обслуживание очередного прерывания происходит циклический сдвиг приоритетов.

4. Заданный сдвиг приоритетов по команде программиста.

Программирование контроллера.

Для программирования контроллеров прерываний используется два регистра. По два на каждый контроллер. Для первого контроллера 20H и 21H, для второго контроллера A0H и A1H.

1. Команды инициализаций. Инициализация должна производиться над обоими контроллерами.

Команда 1. Формат засылаемого байта:
Бит 0 = 1,
Бит 1 - для двух контроллеров значение этого бита должно быть равно О,
Бит 2 = О,
Бит 3 - для стандартных AT компьютеров этот бит равен О,
Бит4=1,
Бит5 = 0,
Бит6 = 0,
Бит 7 = 0.
Выдается по адресу 20H (AOH).

Команда 2. Формат засылаемого байта:
Биты О-2 = 0,
Биты 3-7 - определяют 5 старших битов номеров Прерываний для всех линий контроллера (базовый вектор).
Выдается по адресу 21H (A1H).

Команда 3. Формат посылаемого байта:
Для ведущего контроллера - 00000100,
Для ведомого контроллера - 00000010.
Выдается по адресу 21Н (А1Н). Только при наличии двух контроллеров.

Команда 4. Формат посылаемого байта:
Бит 0 - 1 (совместимость с процессором Intel),
Бит 1 - 0(обычно 1 - режим автоматического завершения прерывания),
Бит 2 - рекомендуется 0,
Бит 3 - рекомендуется О,
Бит 4 - 0 - режим полной вложенности.
Выдается по адресу 21Н (А1Н).

2. Другие команды контроллера.

Изменение регистра маски (IMR). Выдается по адресу 21Н (А1Н).

Завершение процедуры обработки прерывания контроллером. Формат посылаемого байта:
Биты 0-2 - задают номер линии, когда это требуется. Иначе следует обнулять,
Бит3 = 0,
Бит4 = 0,
Биты 5-7;
001 - команда конца прерывания 1 (см. выше),
011 - команда конца прерывания 2,
101 - автоматический сдвиг приоритетов,
100 - установка сдвига приоритетов,
000 - сброс сдвига,
111 - сдвиг по команде,
110 - установка приоритетов (номер линии в первых битах),
010 - нет операции.
Выдается по адресу 20Н (АОН).

Команда изменения режима. Формат посылаемого байта:
Биты 0-1:
10 - считать регистр IRR,
11 - считать регистр ISR.
Бит 2 = 0,
Бит3 = 1,
Бит4 = 0,
Биты 5-6:
10 - специальный режим маскирования,
11 - сбросить специальный режим маскирования.
Выдается по адресу 20Н (АОН). Чтение регистров производится по этому же адресу.

Другие порты.

Порт с адресом **61Н.** Доступен по записи и чтению.

Управляющий регистр В.

Структура для чтения:

Бит 0 - 0 сигнал GATE установлен в низкий уровень (линия GATE второго канала таймера), 1 - в высокий,

бит 1 - 0 - второй канал таймера отключен от динамика, 1 - подключен,

бит 2 - 0 - прерывание по ошибке четности памяти разрешено, 1 - запрещено,

бит 3 - 0 - прерывание по ошибке канала разрешено, 1 - запрещено,

бит 4 - изменяет свое значение на каждом цикле регенерации памяти,

бит 5 - определяет состояние линии OUT канала 2 таймера,

бит 6 - 0 - нет ошибки канала, 1 - ошибка,

бит 7 - 0 - нет ошибки четности, 1 - ошибка.

Структура для записи:

бит 0 - установить сигнал GATE в низкий уровень, 1 - в высокий,

бит 1 - 0 - отключить канал 2 от динамика, 1 - подключить,

бит 2 - 0 - прерывание по ошибке четности памяти разрешено, 1 - запрещено,

бит 3 - 0 - прерывание по ошибке канала разрешено, 1 - запрещено,

бит 4-6 - резерв,

бит 7 - недействителен.

Литература.

Здесь приводится достаточно полный список литературы, которую я бы порекомендовал всем начинающим программировать на языке ассемблера. Некоторые источники дошли до меня в виде файлов, и все же я хотел бы сослаться на них, как на очень добротную и полезную литературу по программированию на IBM PC. В списке такая литература отмечена значком (F). Особо отмечу, что если для MS DOS литература по программированию на языке ассемблера весьма обширна, то литература по программированию на ассемблере для Windows отсутствует вообще[70]. Вам придется пользоваться книжками по Си. В конце списка приведены, на мой взгляд, наиболее полезные из таких книжек.

1. С.П. МОРС, Д.Д. АЛБЕРТ. Архитектура микропроцессора 80286. М., 1990.
2. В.Л. ГРИГОРЬЕВ. Архитектура и программирование арифметического сопроцессора. М., 1991.
3. Л. СКЭНЛОН. Персональные ЭВМ IBM PC и XT. Программирование на языке ассемблера. М.,1991.
4. П. АБЕЛЬ. Язык ассемблера для IBM PC и программирования. М.,1992.
5. Р. Журден. Справочник программиста персональных компьютеров типа IBM PCXT и AT. М., 1992.
6. Ю-ЧЖЕН ЛЮ, Г. ГИБСОН. Микропроцессоры семейства 8086/8088. М., 1987.
7. Этот безумный, безумный, безумный мир резидентных программ. В "Компьютер пресс", № 4, 5, 1992.
8. RICHARD WILTON. PROGRAMMER'S GUID TO PC & PS/2 VIDEO SYSTEMS. MAXIMUM VIDEO. PERFORMANCE FROM THE EGA, VGA, HGC AND MCGA. MICROSOFT PRESS, 1987, WASHINGTON.
9. П. Нортон. Персональный компьютер фирмы IBM и операционная система MS DOS. М., 1991.
10. П. Нортон, Д. Соухэ. Язык ассемблера для IBM PC. М., 1993.
11. MICROSOFT. MS-DOS. PROGRAMMER'S REFERENCE. VERSION 4.X. MICROSOFT PRESS, 1988, WASHINGTON. (F).
12. В.Ю. Романов. Популярные форматы файлов для хранения графических изображений. М., 1992.
13. Tech Help. The Electronic Technical Reference Manuel. By Dan Rollins. 1990. (F).
14. А. Щербаков Защита от копирования. М., 1992.
15. А.В. Спесивцев, В.А. Вегнер и др. Защита информации в персональных ЭВМ. М., 1992.
16. Смешанное программирование на языках Бейсик, Паскаль, Си, Ассемблер (фирмы Микрософт, USA). Руководство программиста. 1990. Перевод документа "Mixed-Language Programming Guide", выпущенного фирмой Microsoft. (F).

---

[70] В последнее время в Интернете появились отдельные статьи по программированию на ассемблере в операционной системе Windows.

17. К. Брэдли. Программирование на языке ассемблера для персональных компьютеров фирмы IBM.
18. Э. Страусе. Микропроцессор 80286. Versus Ltd., М., 1992.
19. В. Фролов, Г.В. Фролов. Защищенный режим процессоров Intel 80286/80386/80486. Москва, 1993.
20. Л.В. Лямлин. Макроассемблер MASM. М.,1994.
21. В.Л. Григорьев. Видеосистемы ПК фирмы IBM. М., 1993.
22. В.Л. Григорьев. Микропроцессор i486. Архитектура и программирование (в 4 книгах). М., 1993.
23. Р. Лэй. Разработка драйверов устройств для MS DOS. Рязань, 1992.
24. Т. Хоган. Аппаратные и программные средства персональных компьютеров. Справочник, Ч. 1,2. М., 1995.
25. В.А. Вегнер, А.Ю. Крутяков и др. Аппаратура персональных компьютеров и ее программирование. Москва, 1995.
26. С. Лукач, А. Е. Сибиряков. Программно-технические средства персональных ЭВМ семейства IBM PC. Свердловск, 1990. (F).
27. С.А. Гладков, Г.В. Фролов. Программирование в Microsoft Windows. В двух частях. Москва, 1992.
28. Герберт Шилдт. Программирование на С и С++ для Windows 95. Киев. 1996.
29. Мэтт Питрек. Секреты системного программирования в Windows 95. Киев. 1996.
30. Барри Нанс. Программирование в локальных сетях. М., 1990.
31. А.В. Фролов, Г.В. Фролов. Локальные сети персональных компьютеров (в 3 частях) М.,1993.
32. Михаил Гук. Процессоры INTEL от 8086 до Pentium П. Санкт-Петербург, 1998.

Алфавитный указатель.

Алфавитный указатель дополняет оглавление. Приводятся ссылки не на все упоминания, а только на те, которые раскрывают суть данного термина.